# Contents

<table>
<thead>
<tr>
<th>Section number</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 1</td>
<td>Introduction</td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>Release Notes</td>
<td>13</td>
</tr>
<tr>
<td>1.2</td>
<td>About this Guide</td>
<td>13</td>
</tr>
<tr>
<td>1.3</td>
<td>Accompanying Documentation</td>
<td>14</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>Getting Started</td>
<td></td>
</tr>
<tr>
<td>2.1</td>
<td>About CodeWarrior Interface</td>
<td>15</td>
</tr>
<tr>
<td>2.2</td>
<td>About Data Collection</td>
<td>17</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Trace Data</td>
<td>17</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Timeline Data</td>
<td>17</td>
</tr>
<tr>
<td>2.2.3</td>
<td>Critical Code Data</td>
<td>18</td>
</tr>
<tr>
<td>2.2.4</td>
<td>Performance Data</td>
<td>18</td>
</tr>
<tr>
<td>2.2.5</td>
<td>Call Tree Data</td>
<td>18</td>
</tr>
<tr>
<td>2.2.6</td>
<td>Counterpoints</td>
<td>18</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>Collecting Data</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Process for Collecting Data</td>
<td>21</td>
</tr>
<tr>
<td>3.2</td>
<td>Creating New Project</td>
<td>22</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Simulator Profiling</td>
<td>22</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Hardware Profiling</td>
<td>27</td>
</tr>
<tr>
<td>3.3</td>
<td>Configuring Launcher</td>
<td>33</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Simulator Profiling</td>
<td>33</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Hardware Profiling</td>
<td>38</td>
</tr>
<tr>
<td>3.4</td>
<td>Collecting Trace</td>
<td>46</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Using Simulator</td>
<td>47</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Using Hardware</td>
<td>48</td>
</tr>
<tr>
<td>Section number</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.5</td>
<td>Viewing Data</td>
<td>49</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Viewing Data Generated by Simulator</td>
<td>50</td>
</tr>
<tr>
<td>3.5.1.1</td>
<td>Viewing Trace Data</td>
<td>50</td>
</tr>
<tr>
<td>3.5.1.1.1</td>
<td>Exporting Trace Data</td>
<td>57</td>
</tr>
<tr>
<td>3.5.1.1.2</td>
<td>Configuring Time Unit and Time Format</td>
<td>58</td>
</tr>
<tr>
<td>3.5.1.2</td>
<td>Viewing Timeline Data</td>
<td>59</td>
</tr>
<tr>
<td>3.5.1.2.1</td>
<td>Add or Remove Function</td>
<td>62</td>
</tr>
<tr>
<td>3.5.1.2.2</td>
<td>Edit Address Range of Function</td>
<td>62</td>
</tr>
<tr>
<td>3.5.1.2.3</td>
<td>Change Color</td>
<td>62</td>
</tr>
<tr>
<td>3.5.1.2.4</td>
<td>Add or Remove Group</td>
<td>62</td>
</tr>
<tr>
<td>3.5.1.2.5</td>
<td>Merge Groups or Functions</td>
<td>64</td>
</tr>
<tr>
<td>3.5.1.3</td>
<td>Viewing Critical Code Data</td>
<td>65</td>
</tr>
<tr>
<td>3.5.1.3.1</td>
<td>Export Critical Code Data</td>
<td>69</td>
</tr>
<tr>
<td>3.5.1.4</td>
<td>Viewing Performance Data</td>
<td>72</td>
</tr>
<tr>
<td>3.5.1.5</td>
<td>Viewing Call Tree Data</td>
<td>75</td>
</tr>
<tr>
<td>3.5.1.6</td>
<td>Viewing Counterpoints</td>
<td>77</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Viewing Data Generated by Hardware</td>
<td>77</td>
</tr>
<tr>
<td>3.5.2.1</td>
<td>Viewing Trace Data</td>
<td>78</td>
</tr>
<tr>
<td>3.5.2.2</td>
<td>Viewing Timeline Data</td>
<td>79</td>
</tr>
<tr>
<td>3.5.2.3</td>
<td>Viewing Critical Code Data</td>
<td>79</td>
</tr>
<tr>
<td>3.5.2.4</td>
<td>Viewing Performance Data</td>
<td>80</td>
</tr>
<tr>
<td>3.5.2.5</td>
<td>Viewing Counterpoints</td>
<td>80</td>
</tr>
<tr>
<td>3.6</td>
<td>Exclude Symbols from Statistics</td>
<td>80</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Software Analysis View</td>
<td>80</td>
</tr>
<tr>
<td>3.6.2</td>
<td>Critical Code View</td>
<td>85</td>
</tr>
<tr>
<td>3.6.3</td>
<td>Performance View</td>
<td>86</td>
</tr>
<tr>
<td>3.6.4</td>
<td>Call Tree View</td>
<td>87</td>
</tr>
</tbody>
</table>
### Section number | Title | Page
---|---|---
3.7 | Trace on Attach | 88
  3.7.1 | On Application Crash Investigation | 88
    3.7.1.1 | Create an Attach Launch Configuration | 89
    3.7.1.2 | Upload Trace | 90
    3.7.1.3 | Investigate Crash | 90
  3.7.2 | On Changing Trace Configuration | 92
    3.7.2.1 | Create an Attach Launch Configuration | 93
    3.7.2.2 | Change Trace Configuration | 93
3.8 | Viewing Interrupts | 93
3.9 | Preparing Aurora Interface for Collecting Aurora Nexus Trace | 95
  3.9.1 | Enabling Aurora | 95
  3.9.2 | Executing Aurora Training Script | 95
3.10 | Inline Functions | 97
3.11 | Importing Offline SC3900 Trace with Hardware Trace Configuration | 101
3.12 | Multicore Tracing | 109
  3.12.1 | Debugging Multicore Project | 110
  3.12.2 | Collecting Multicore Trace Data | 114
  3.12.3 | Viewing multicore trace data | 114
  3.12.4 | Importing multicore trace data | 119
  3.12.5 | Trace commander view | 120
  3.12.6 | Multicore continuous trace | 123

---

**Chapter 4**

**Setting Tracepoints**

4.1 | Setting Start and Stop Tracepoints | 125
  4.1.1 | Simulator | 126
  4.1.2 | Target Hardware | 130
    4.1.2.1 | Modifying Hardware Tracepoint Configuration | 134
  4.1.3 | Viewing Tracepoints in Analysispoints | 135
    4.1.3.1 | Disassembly View | 135
<table>
<thead>
<tr>
<th>Section number</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.3.2</td>
<td>Editor Area</td>
<td>136</td>
</tr>
<tr>
<td>4.2</td>
<td>Enabling and Disabling the Tracepoints</td>
<td>138</td>
</tr>
<tr>
<td>4.3</td>
<td>Limitation of hardware tracepoints</td>
<td>138</td>
</tr>
<tr>
<td>4.4</td>
<td>Multicore hardware tracepoints</td>
<td>140</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Set cores</td>
<td>140</td>
</tr>
<tr>
<td>5.1</td>
<td>Setting Counterpoints</td>
<td>143</td>
</tr>
<tr>
<td>5.2</td>
<td>Viewing Counterpoints in Analysispoints</td>
<td>149</td>
</tr>
<tr>
<td>5.3</td>
<td>Enabling and Disabling the Counterpoints</td>
<td>149</td>
</tr>
<tr>
<td>6.1</td>
<td>Performance Analysis Perspective</td>
<td>151</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Performance Analysis View</td>
<td>152</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Configuration View</td>
<td>153</td>
</tr>
<tr>
<td>6.1.2.1</td>
<td>Connection Pane</td>
<td>154</td>
</tr>
<tr>
<td>6.1.2.2</td>
<td>Scenarios Pane</td>
<td>155</td>
</tr>
<tr>
<td>6.1.2.3</td>
<td>Sampling Pane</td>
<td>156</td>
</tr>
<tr>
<td>6.1.2.4</td>
<td>Filter Core Events pane</td>
<td>157</td>
</tr>
<tr>
<td>6.1.2.5</td>
<td>Target Information Pane</td>
<td>158</td>
</tr>
<tr>
<td>6.1.2.6</td>
<td>Session pane</td>
<td>159</td>
</tr>
<tr>
<td>6.1.2.7</td>
<td>Realtime Visualization</td>
<td>160</td>
</tr>
<tr>
<td>6.1.3</td>
<td>Performance Analysis Session View</td>
<td>160</td>
</tr>
<tr>
<td>6.1.4</td>
<td>Progress View</td>
<td>161</td>
</tr>
<tr>
<td>6.2</td>
<td>Creating New Configuration</td>
<td>162</td>
</tr>
<tr>
<td>6.3</td>
<td>Connecting to Board</td>
<td>164</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Adding Connection Manually</td>
<td>165</td>
</tr>
<tr>
<td>6.4</td>
<td>Selecting Scenarios</td>
<td>167</td>
</tr>
<tr>
<td>6.5</td>
<td>Running Scenarios</td>
<td>170</td>
</tr>
<tr>
<td>Section number</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>6.6</td>
<td>Viewing Results</td>
<td>171</td>
</tr>
<tr>
<td>6.6.1</td>
<td>Tabular Format</td>
<td>172</td>
</tr>
<tr>
<td>6.6.2</td>
<td>Graphical Format</td>
<td>173</td>
</tr>
<tr>
<td>6.7</td>
<td>Custom scenarios</td>
<td>176</td>
</tr>
<tr>
<td>6.7.1</td>
<td>Defining custom scenarios</td>
<td>176</td>
</tr>
<tr>
<td>6.7.2</td>
<td>Editing custom scenarios</td>
<td>177</td>
</tr>
<tr>
<td>6.7.2.1</td>
<td>Overview</td>
<td>177</td>
</tr>
<tr>
<td>6.7.2.2</td>
<td>Events</td>
<td>178</td>
</tr>
<tr>
<td>6.7.2.3</td>
<td>Metrics</td>
<td>179</td>
</tr>
<tr>
<td>6.7.2.4</td>
<td>Initialization</td>
<td>181</td>
</tr>
<tr>
<td>6.7.2.5</td>
<td>Reports</td>
<td>182</td>
</tr>
<tr>
<td></td>
<td>Chapter 7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Launching Scripts</td>
<td></td>
</tr>
<tr>
<td>7.1</td>
<td>Run Sample Python Script</td>
<td>185</td>
</tr>
<tr>
<td>7.2</td>
<td>Collect Trace Using Jython</td>
<td>186</td>
</tr>
<tr>
<td></td>
<td>Chapter 8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Remote Launch API</td>
<td></td>
</tr>
<tr>
<td>8.1</td>
<td>com::freescale::sa::scripting::api::AnalysisActions</td>
<td>189</td>
</tr>
<tr>
<td>8.1.1</td>
<td>Public Member Functions</td>
<td>189</td>
</tr>
<tr>
<td>8.1.1.1</td>
<td>AnalysisActions</td>
<td>190</td>
</tr>
<tr>
<td>8.1.1.2</td>
<td>startTrace</td>
<td>190</td>
</tr>
<tr>
<td>8.1.1.3</td>
<td>stopTrace</td>
<td>190</td>
</tr>
<tr>
<td>8.1.2</td>
<td>Public Static Member Functions</td>
<td>190</td>
</tr>
<tr>
<td>8.1.2.1</td>
<td>startTrace</td>
<td>191</td>
</tr>
<tr>
<td>8.1.2.2</td>
<td>stopTrace</td>
<td>191</td>
</tr>
<tr>
<td>8.2</td>
<td>com::freescale::sa::scripting::api::AnalysisConfig</td>
<td>191</td>
</tr>
<tr>
<td>8.2.1</td>
<td>Protected Member Functions</td>
<td>191</td>
</tr>
<tr>
<td>8.2.1.1</td>
<td>AnalysisConfig</td>
<td>192</td>
</tr>
<tr>
<td>8.2.1.2</td>
<td>setProject</td>
<td>192</td>
</tr>
</tbody>
</table>

CodeWarrior Development Studio for StarCore SC3900FP DSP Architectures Tracing and Analysis Tools
User Guide, Rev. 10.9.0, 11/2015
Freescale Semiconductor, Inc.
<table>
<thead>
<tr>
<th>Section number</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.2.1.3</td>
<td>load</td>
<td>192</td>
</tr>
<tr>
<td>8.2.1.4</td>
<td>save</td>
<td>192</td>
</tr>
<tr>
<td>8.2.1.5</td>
<td>getWorkingCopy</td>
<td>192</td>
</tr>
<tr>
<td>8.2.1.6</td>
<td>getLaunchConfiguration</td>
<td>193</td>
</tr>
<tr>
<td>8.2.2</td>
<td>Public Member Functions</td>
<td>193</td>
</tr>
<tr>
<td>8.2.2.1</td>
<td>getOutputFolder</td>
<td>193</td>
</tr>
<tr>
<td>8.2.2.2</td>
<td>addSrcLineTracePoint</td>
<td>194</td>
</tr>
<tr>
<td>8.2.2.3</td>
<td>addAddressTracePoint</td>
<td>194</td>
</tr>
<tr>
<td>8.2.2.4</td>
<td>removeTracePoint</td>
<td>195</td>
</tr>
<tr>
<td>8.2.2.5</td>
<td>getTracePoints</td>
<td>195</td>
</tr>
<tr>
<td>8.2.2.6</td>
<td>setIgnoreAllTracePoints</td>
<td>196</td>
</tr>
<tr>
<td>8.2.2.7</td>
<td>getIgnoreAllTracePoints</td>
<td>196</td>
</tr>
<tr>
<td>8.2.2.8</td>
<td>removeAllTracePoints</td>
<td>196</td>
</tr>
<tr>
<td>8.2.2.9</td>
<td>removeLineCounterpoint</td>
<td>196</td>
</tr>
<tr>
<td>8.2.2.10</td>
<td>removeAddrCounterpoint</td>
<td>197</td>
</tr>
<tr>
<td>8.2.2.11</td>
<td>setLineCounterpointEnabled</td>
<td>197</td>
</tr>
<tr>
<td>8.2.2.12</td>
<td>setAddrCounterpointEnabled</td>
<td>198</td>
</tr>
<tr>
<td>8.2.2.13</td>
<td>getAttribute</td>
<td>198</td>
</tr>
<tr>
<td>8.2.2.14</td>
<td>setAttribute</td>
<td>198</td>
</tr>
<tr>
<td>8.2.3</td>
<td>Public Static Member Functions</td>
<td>199</td>
</tr>
<tr>
<td>8.2.3.1</td>
<td>removeAllTracepointsByType</td>
<td>199</td>
</tr>
<tr>
<td>8.3</td>
<td>com::freescale::sa::scripting::api::AnalysisErrors</td>
<td>199</td>
</tr>
<tr>
<td>8.3.1</td>
<td>Public Static Member Functions</td>
<td>200</td>
</tr>
<tr>
<td>8.3.1.1</td>
<td>log</td>
<td>200</td>
</tr>
<tr>
<td>8.4</td>
<td>com::freescale::sa::scripting::api::AnalysisFactory</td>
<td>200</td>
</tr>
<tr>
<td>8.4.1</td>
<td>Public Static Member Functions</td>
<td>200</td>
</tr>
<tr>
<td>8.4.1.1</td>
<td>getAnalysisConfig</td>
<td>201</td>
</tr>
<tr>
<td>8.4.1.2</td>
<td>getAnalysisConfigFromLaunch</td>
<td>201</td>
</tr>
<tr>
<td>8.4.1.3</td>
<td>getAnalysisResults</td>
<td>201</td>
</tr>
<tr>
<td>Section number</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>----------------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>8.4.1.4</td>
<td>getAnalysisResults</td>
<td>202</td>
</tr>
<tr>
<td>8.4.1.5</td>
<td>getAnalysisResults</td>
<td>202</td>
</tr>
<tr>
<td>8.4.1.6</td>
<td>getAnalysisResults</td>
<td>203</td>
</tr>
<tr>
<td>8.5</td>
<td>com::freescale::sa::scripting::api::AnalysisResults</td>
<td>203</td>
</tr>
<tr>
<td>8.5.1</td>
<td>Protected Member Functions</td>
<td>204</td>
</tr>
<tr>
<td>8.5.1.1</td>
<td>AnalysisResults</td>
<td>204</td>
</tr>
<tr>
<td>8.5.1.2</td>
<td>AnalysisResults</td>
<td>204</td>
</tr>
<tr>
<td>8.5.1.3</td>
<td>exportTraceData</td>
<td>205</td>
</tr>
<tr>
<td>8.5.2</td>
<td>Public Member Functions</td>
<td>205</td>
</tr>
<tr>
<td>8.5.2.1</td>
<td>getTraceRecordsNo</td>
<td>205</td>
</tr>
<tr>
<td>8.5.2.2</td>
<td>exportResults</td>
<td>205</td>
</tr>
<tr>
<td>8.5.2.3</td>
<td>exportFunctionResults</td>
<td>206</td>
</tr>
<tr>
<td>8.5.3</td>
<td>Public Static Member Functions</td>
<td>206</td>
</tr>
<tr>
<td>8.5.3.1</td>
<td>logMessages</td>
<td>207</td>
</tr>
<tr>
<td>8.6</td>
<td>com::freescale::sa::sc::scripting::api::ScAnalysisConfig</td>
<td>207</td>
</tr>
<tr>
<td>8.6.1</td>
<td>Protected Member Functions</td>
<td>207</td>
</tr>
<tr>
<td>8.6.1.1</td>
<td>ScAnalysisConfig</td>
<td>207</td>
</tr>
<tr>
<td>8.6.1.2</td>
<td>ScAnalysisConfig</td>
<td>207</td>
</tr>
<tr>
<td>8.6.2</td>
<td>Public Member Functions</td>
<td>208</td>
</tr>
<tr>
<td>8.6.2.1</td>
<td>getAttribute</td>
<td>208</td>
</tr>
<tr>
<td>8.6.2.2</td>
<td>setAttribute</td>
<td>208</td>
</tr>
<tr>
<td>8.7</td>
<td>com::freescale::sa::sc::scripting::api::ScHwAnalysisActions</td>
<td>209</td>
</tr>
<tr>
<td>8.7.1</td>
<td>Public Member Functions</td>
<td>209</td>
</tr>
<tr>
<td>8.7.1.1</td>
<td>ScHwAnalysisActions</td>
<td>209</td>
</tr>
<tr>
<td>8.7.1.2</td>
<td>uploadTrace</td>
<td>209</td>
</tr>
<tr>
<td>8.7.2</td>
<td>Public Static Member Functions</td>
<td>210</td>
</tr>
<tr>
<td>8.7.2.1</td>
<td>uploadTrace</td>
<td>210</td>
</tr>
<tr>
<td>Section number</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>8.8</td>
<td>com::freescale::sa::sc::scripting::api::ScHwAnalysisConfig</td>
<td>210</td>
</tr>
<tr>
<td>8.8.1</td>
<td>Public Member Functions</td>
<td>210</td>
</tr>
<tr>
<td>8.8.1.1</td>
<td>ScHwAnalysisConfig</td>
<td>211</td>
</tr>
<tr>
<td>8.8.1.2</td>
<td>ScHwAnalysisConfig</td>
<td>211</td>
</tr>
<tr>
<td>8.8.1.3</td>
<td>setAttribute</td>
<td>211</td>
</tr>
<tr>
<td>8.8.1.4</td>
<td>getAttribute</td>
<td>212</td>
</tr>
<tr>
<td>8.8.2</td>
<td>Protected Member Functions</td>
<td>212</td>
</tr>
<tr>
<td>8.8.2.1</td>
<td>setUserDefinedEventsTrace</td>
<td>213</td>
</tr>
<tr>
<td>8.8.2.2</td>
<td>isUserDefinedEventsTrace</td>
<td>213</td>
</tr>
<tr>
<td>8.8.2.3</td>
<td>setOwnershipTrace</td>
<td>213</td>
</tr>
<tr>
<td>8.8.2.4</td>
<td>isOwnershipTrace</td>
<td>213</td>
</tr>
<tr>
<td>8.8.2.5</td>
<td>setTraceScenario</td>
<td>214</td>
</tr>
<tr>
<td>8.8.2.6</td>
<td>getTraceScenario</td>
<td>214</td>
</tr>
<tr>
<td>8.8.2.7</td>
<td>load</td>
<td>214</td>
</tr>
<tr>
<td>8.9</td>
<td>com::freescale::sa::sc::scripting::api::ScSimAnalysisConfig</td>
<td>215</td>
</tr>
<tr>
<td>8.9.1</td>
<td>Public Member Functions</td>
<td>215</td>
</tr>
<tr>
<td>8.9.1.1</td>
<td>ScSimAnalysisConfig</td>
<td>215</td>
</tr>
<tr>
<td>8.9.1.2</td>
<td>ScSimAnalysisConfig</td>
<td>215</td>
</tr>
<tr>
<td>8.9.1.3</td>
<td>setAttribute</td>
<td>216</td>
</tr>
<tr>
<td>8.9.1.4</td>
<td>getAttribute</td>
<td>216</td>
</tr>
<tr>
<td>8.9.2</td>
<td>Protected Member Functions</td>
<td>216</td>
</tr>
<tr>
<td>8.9.2.1</td>
<td>load</td>
<td>217</td>
</tr>
<tr>
<td>8.9.2.2</td>
<td>setCommPortNumber</td>
<td>217</td>
</tr>
<tr>
<td>8.9.2.3</td>
<td>getCommPortNumber</td>
<td>217</td>
</tr>
<tr>
<td>8.9.2.4</td>
<td>setOutputFolder</td>
<td>218</td>
</tr>
<tr>
<td>8.9.2.5</td>
<td>enableLogging</td>
<td>218</td>
</tr>
<tr>
<td>8.9.2.6</td>
<td>isLoggingEnabled</td>
<td>218</td>
</tr>
<tr>
<td>Section number</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>8.10</td>
<td>com::freescale::sa::scripting::api::TracePoint</td>
<td>218</td>
</tr>
<tr>
<td></td>
<td>8.10.1 Protected Member Functions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8.10.1.1 TracePoint</td>
<td>219</td>
</tr>
<tr>
<td></td>
<td>8.10.1.2 getITracepoint</td>
<td>219</td>
</tr>
<tr>
<td></td>
<td>8.10.2 Public Member Functions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8.10.2.1 setEnabled</td>
<td>220</td>
</tr>
<tr>
<td></td>
<td>8.10.2.2 getEnabled</td>
<td>220</td>
</tr>
<tr>
<td></td>
<td>8.10.2.3 getAction</td>
<td>220</td>
</tr>
<tr>
<td></td>
<td>8.10.2.4 getType</td>
<td>220</td>
</tr>
<tr>
<td></td>
<td>8.10.2.5 getAddress</td>
<td>220</td>
</tr>
<tr>
<td></td>
<td>8.10.2.6 getFileName</td>
<td>221</td>
</tr>
<tr>
<td></td>
<td>8.10.2.7 getLineNumber</td>
<td>221</td>
</tr>
<tr>
<td>8.11</td>
<td>com.freescale.sa.scripting.IAnalysisConfigFactory</td>
<td>221</td>
</tr>
<tr>
<td></td>
<td>8.11.1 Public Member Functions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8.11.1.1 createAnalysisConfigFromArchive</td>
<td>222</td>
</tr>
<tr>
<td></td>
<td>8.11.1.2 createAnalysisConfigFromLaunch</td>
<td>222</td>
</tr>
</tbody>
</table>
Chapter 1
Introduction

The CodeWarrior Tracing and Analysis tool set to develop software for Freescale StarCore DSP processors.

This chapter explains:

- Release Notes
- About this Guide
- Accompanying Documentation

1.1 Release Notes

Release notes include information about new features, last-minute changes, bug fixes, incompatible elements, or other sections that may not be included in this manual.

You should read release notes before using the CodeWarrior IDE.

NOTE

The release notes for specific components of the CodeWarrior IDE are located in the Release_Notes folder in the CodeWarrior installation directory.

1.2 About this Guide

Each chapter of this manual describes a different area of software development.
The table below lists each chapter in the manual.

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>(this chapter)</td>
</tr>
<tr>
<td>Getting Started</td>
<td>Describes the analysis tool and the trace and profiling data</td>
</tr>
<tr>
<td>Collecting Data</td>
<td>Describes how to collect the data from simulator SC3900 and Qonverge targets</td>
</tr>
<tr>
<td>Setting Tracepoints</td>
<td>Describes how to collect selective data after setting the tracepoints</td>
</tr>
<tr>
<td>Setting Counterpoints</td>
<td>Describes how to count events between two points of the executed code</td>
</tr>
<tr>
<td>Performance Analysis for B4 Devices</td>
<td>Describes how to configure, collect, and analyze scenarios in the B4 devices.</td>
</tr>
<tr>
<td>Launching Scripts</td>
<td>Describes how to collect and export trace data automatically using remote launch.</td>
</tr>
<tr>
<td>Remote Launch API</td>
<td>Lists the classes and interface for the remote launch API.</td>
</tr>
</tbody>
</table>

1.3 Accompanying Documentation

The Documentation Roadmap page describes the documentation included in this version of CodeWarrior Development Studio for StarCore DSP Architectures.

You access Documentation Roadmap by:

- a shortcut link on the Desktop that the installer creates by default, or
- opening START_HERE.html in CWInstallDir\SC\Help.
Chapter 2
Getting Started

CodeWarrior Tracing and Analysis Tools provide visibility into an application as it runs on the simulator SC3900 and the Qonverge targets. This visibility can help you understand how your application runs, as well as identify operational problems.

Tracing and analysis results are accurate for O0 optimization level. For an optimization level higher than O0, the results that are provided by this tool can be affected. The affected results depend on the type of code and its optimization, whereas the results that are based on asm code, analysis are not affected by optimization level. The results based on source code and name of symbols can be affected by optimization level. For an accurate analysis of the libraries linked to your project, make sure to generate debugging information for them so that all the debug symbols are found and included in the statistics.

However, it is possible to obtain limited accuracy only for profiling trace(Trace, Timeline, Performance and Call Tree results) on O3 optimization level independent of the type of code. The limited accuracy is defined due to the following reasons, when:

- There is no debug information generated for inlined functions
- The compiler has tail call optimization enabled by default, which means that a jsr is transformed into a jmp (bsr vs. bra). It is possible to disable this optimization using the compiler option -Xllt --tailjsr2jmp=0

This chapter explains:

- About CodeWarrior Interface
- About Data Collection

2.1 About CodeWarrior Interface
The CodeWarrior Development Studio provides a common interface for developing, debugging, and analyzing your applications. The project-oriented Workbench window provides numerous perspectives containing views, editors, and controls that appear in menus and tool bars.

After creating a project, build your application, define a launch configuration, and then wait for data collection and data display.

![Figure 2-1. Workbench Window](image)

The perspective is a collection of views within the Workbench window used by the Tracing and Analysis tools.

Standard perspective views:

- **CodeWarrior Projects** view provides a hierarchical view of the project resources in the Workbench. Right-click context menus provide file-management controls.
- **Console** view shows the process output including actions, messages, and errors. It display messages indicating when data collection is enabled, in process, and complete.
- **Software Analysis** view provides a hierarchical view of the project's data sources, data files, and data sets of the simulator.
2.2 About Data Collection

This section lists the types of data collected for an application when it runs on the simulator SC3900 and Qonverge targets.

You can collect the following types of data for an application:

- Trace Data
- Timeline Data
- Critical Code Data
- Performance Data
- Call Tree Data
- Counterpoints

2.2.1 Trace Data

The Trace viewer displays the trace data collected by the simulator and the targets. The features available for the Trace viewer include:

- Export trace data to an Excel file
- Configure Timestamp column of the trace data as absolute or delta values in decimal or hexadecimal format
- Set CPU frequency and convert the clock cycles into real time in milliseconds, microseconds, or nanoseconds, displayed in the Timestamp column
- Copy and paste a cell or a line of the trace data
- Auto resize the column width or a row
- Quick navigation through trace using navigation buttons
- Expand the trace lines for more detailed information on the events
- Search capability

2.2.2 Timeline Data

The timeline data displays a graphical view of the functions that are executed in the application and the number of cycles each function takes when the application is run.
2.2.3 Critical Code Data

The critical code data is generated based on the trace data. The Critical Code Data viewer displays name, start address, number of times each instruction is executed, and code size of each function in the program. The Critical Code Data viewer displays the detailed information of every instruction traced in the data.

The features available for the Critical Code Data viewer include:

- statistics at function level,
- statistics at instruction level,
- code view in source editor, and
- column ordering and sorting at function level.

2.2.4 Performance Data

The performance data includes the metric and invocation information for each function that executes in the application. The performance data during measurement enables you to compare the relative efficiencies of various portions of your target program. Both exclusive and inclusive timing measurements are provided in the performance data. The parent-child calling relationships between your program’s functions are also provided. Each function pair consists of a caller and a callee with data provided for each.

2.2.5 Call Tree Data

The Call Tree data shows the general application flow in a hierarchical tree in which statistics are displayed for each function.

2.2.6 Counterpoints
Counterpoints allow you to count events between two points of the executed code. The results are computed based on the collected trace data; the counterpoints are searched in trace by their addresses. When counterpoints are found in the executed code, the delta statistics is computed between two counterpoints.
Chapter 3
Collecting Data

This chapter explains how the program runs on the simulator and Qonverge targets to collect the data.

This chapter describes:

• Process for Collecting Data
• Creating New Project
• Configuring Launcher
• Collecting Trace
• Viewing Data
• Exclude Symbols from Statistics
• Trace on Attach
• Viewing Interrupts
• Preparing Aurora Interface for Collecting Aurora Nexus Trace
• Inline Functions
• Importing Offline SC3900 Trace with Hardware Trace Configuration
• Multicore Tracing

3.1 Process for Collecting Data

This section describes the process for setting up the tools for data collection.

To collect data:

• Create and configure a project for the simulator
• Set up the debugger launch configuration to collect the analysis data from the simulator
• Run the application on the simulator to collect data
3.2 Creating New Project

This section explains the process to create new projects for both simulator and hardware profiling.

You can use the CodeWarrior Bareboard Project Wizard to create new projects.

In this topic:

- Simulator Profiling
- Hardware Profiling

3.2.1 Simulator Profiling

This section describes the process to create a simulator project for collecting trace data.

You can use the CodeWarrior Bareboard Project Wizard to create new projects for tracing.

The CodeWarrior IDE is a project-oriented interface. You must create a new project or open an existing project before using the Analysis tools.

To create a new project:

   
   The CodeWarrior Bareboard Project Wizard page appears.
Figure 3-1. Create a CodeWarrior Bareboard Project Page

2. On the Create a CodeWarrior Bareboard Project page, enter the name of your project and also specify the location of the project.

The following table describes the StarCore Project settings.

Table 3-1. Create a StarCore Project Page Settings

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Project Name</td>
<td>Enter the name of the project in this text box</td>
</tr>
</tbody>
</table>
| Use default location     | • Checked - Project stores the files required to build the program in the Workbench's current workspace directory  
                           • Unchecked - Project files are located in the directory you specify. Use the Location text box to select the directory |
| Location                 | Specifies the directory that contains the project files. Use the Browse button to navigate to the desired directory. This option is only available when Use default location is unchecked |

3. Click Next.

The Processor page appears.
4. On the **Processor** page, choose the target device for your project. Select **SC3900fp** from **StarCore Family** options.

5. Click **Next**.

The **Debug Target Settings** page appears.
6. Select SC3900PACC board from the list.
7. Click Next.

The Build Settings page appears.
8. Do not change the default settings on the **Build Settings** page.
9. Click **Finish**.

   The project is created in the **CodeWarrior Projects** view. In this view, you can see that the *proj-sim* project is created.
3.2.2 Hardware Profiling

This section describes the process to create a new project for collecting trace data. You must create a new project or open an existing project before using the Analysis tools for hardware profiling.

To create a new project:

2. Enter the name of your project and also specify the location of the project.
3. Click **Next**.

   The **Processor** page appears.
4. For hardware profiling, select the desired target (for example, **B4860**) from **StarCore Family** group.
5. Click **Next**.

   The **Debug Target Settings** page appears. Do not change the default settings of this page.
6. Click Next.

The Build Settings page appears.
7. Do not change the default settings on the **Build Settings** page.
8. Click **Next**.
   
   The **SmartDSP OS** page appears.
9. Select the SmartDSP OS option you want to use.
   • Select No to exclude SmartDSP OS support from your project.
   • Select Yes to include SmartDSP OS support in your project.

10. Click **Finish**.

11. The project is created in the **Projects Explorer** view. In this view, you can see that `sc-b4860` project is created.
12. Select **Project > Build Project** to build your project.

## 3.3 Configuring Launcher

You need to configure the launcher for simulator and hardware profiling, before debugging your application.

In this topic:

- Simulator Profiling
- Hardware Profiling

### 3.3.1 Simulator Profiling

Before debugging your application, you need to configure the launcher for simulator tracing.

To configure the launch configuration:

1. In the **CodeWarrior Projects** view, right-click on the project and select **Debug as > Debug Configurations** from the context menu.

   The **Debug Configuration** dialog box appears.

2. Expand the CodeWarrior group in the tree structure on the left, and select the launch configuration corresponding to the project you are using from the list to create a launch configuration, for example, `proj-sim1_Debug_SC3900_Download_core0`.

3. In the **Main** tab of the **Debug Configurations** dialog box, verify that `proj-sim1` is displayed in the **Project** field. If it does not appear, browse to the project.

4. If the application is not displayed in the **Application** field, click **Search Project** to select the application image.
To configure the launch configuration for measurement of data:

1. Click the **Trace and Profile** tab. This tab has **Overview** page, **Basic** page, **Intermediate** page, **Advanced** and **Miscellaneous** page.

   The **Overview** page displays the flow diagram for collecting trace.

   **NOTE**

   The **Advanced** tab is not applicable for Simulator profiling.
2. In the Basic page, specify the Communication Settings.
Choose the free port number from interval 0-65535 for communication settings. The default port number is 55555. This is used in TCP/IP communication between software analysis and CCSSIM2.

3. In the Intermediate page, specify Trace control settings option.
• **Apply Trace Configuration Settings**
  - **Automatically (When debug session starts)** - The trace collection process is started automatically when the debug session is launched.
  - **Manually (Using trace buttons from Trace Commander)** - This is the default behavior. This option is used when you want to manually control the trace collection. The trace collection process will not be started automatically on debug session launch, but the trace configuration options are applied, when you click on **Trace Start/Stop** button.

• **Upload Trace Configuration Settings**
  - **Automatically (When debug session is suspended)** - This is the default behavior. This option is used to upload the trace data automatically.
  - **Manually (Using trace buttons from Trace Commander)** - This option is used when you want to manually upload the trace data using **Upload Trace** button available in the **Trace Commander** view.

**NOTE**
Trace configuration settings will be disabled during debug session.
4. In the **Miscellaneous** tab, specify the path for storing the trace and profile results. Click **Browse** to select the path for the **Output Folder**.

![Figure 3-16. Trace and Profile Tab - Miscellaneous Page](image)

**NOTE**

Before using a simulator to collect trace data, a configuration must be made. You can create the configuration in the debug mode. If the project is started without a configuration, a warning message will appear when trying to start the trace collection. No trace data will be collected during that session.

### 3.3.2 Hardware Profiling

You need to configure the launch configuration for hardware profiling.

Perform the following steps to configure the launch configuration:

1. In the CodeWarrior Projects view, right-click on the project and select **Debug as > Debug Configurations** from the context-menu.
The Debug Configurations dialog box appears.

2. Select **CodeWarrior Download** in the tree structure on the left and select the launch configuration corresponding to the project you are using from the list to create a launch configuration, for example, `proj-test_Debug_B4860_Download_core0 (0...5)`.

3. In the Main page, verify that `proj-test` is displayed in the **Project** field. If it does not appear, browse to the project.

4. If the application is not displayed in the **C/C++Application** field, click **Search Project** to select the application image.

![Figure 3-17. Debug Configurations Dialog Box - Main Page](image)

To configure the launch configuration for measurement of data:

1. Click the **Trace and Profile** tab. This tab has **Overview** page, **Basic** page, **Intermediate** page, **Advanced** and **Miscellaneous** page.

   The **Overview** page displays the flow diagram for collecting trace.
2. In the **Basic** page, you need to configure platform configuration, trace module and trace scenarios.
The below table lists the basic trace and profile options required for hardware profiling.

### Table 3-2. Trace and Profile Basic Settings

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
</table>
| Platform Configuration   | allows to select the Platform Configuration file from the drop-down list. A platform configuration file holds the settings made in the Trace and Profile tab page for a project. Changes made in the configuration file of a project will automatically get reflected in all the projects sharing the same configuration file. You can also perform the following actions on a configuration file:
  - **New**: Creates a new configuration file with default settings |

*Table continues on the next page...*
### Table 3-2. Trace and Profile Basic Settings (continued)

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Options</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>• Rename:</td>
<td>Renames the currently selected configuration file.</td>
</tr>
<tr>
<td>• Delete:</td>
<td>Deletes the currently selected configuration file.</td>
</tr>
<tr>
<td><strong>Trace module configured by</strong></td>
<td></td>
</tr>
<tr>
<td>User code</td>
<td>allows to perform trace configuration settings from coderun on target</td>
</tr>
<tr>
<td>CodeWarrior</td>
<td>allows to do the trace settings using CodeWarrior configuration windows</td>
</tr>
<tr>
<td><strong>Trace Scenarios</strong></td>
<td></td>
</tr>
<tr>
<td>Profiling - L2 cache events</td>
<td>trace values of counters of both triad A and B on subroutine/interrupt call/return instructions</td>
</tr>
<tr>
<td>Profiling - data loads</td>
<td>trace values of counters of both triad A and B on subroutine/interrupt call/return instructions</td>
</tr>
<tr>
<td>Profiling - clock cycles</td>
<td>trace values of counters of triad A on subroutine/interrupt call/return instructions</td>
</tr>
<tr>
<td>Profiling - advanced</td>
<td>traces each change of flow instructions</td>
</tr>
<tr>
<td>Program trace</td>
<td>allows program trace to be collected</td>
</tr>
<tr>
<td>Coverage</td>
<td>for C source lines, displays the percentage of number of assembly instructions executed from the total number of assembly instructions corresponding to the source line. For assembly source lines, it shows if the instructions were executed or not</td>
</tr>
<tr>
<td>None</td>
<td>no trace will be collected</td>
</tr>
<tr>
<td><strong>Extend Trace Scenario with:</strong></td>
<td></td>
</tr>
<tr>
<td>Ownership Trace</td>
<td>traces information on current task ID. Ownership trace facilitates tracking the active operating system task by providing visibility to the special purpose registers designated for use by the OS for process ID</td>
</tr>
<tr>
<td>User defined events</td>
<td>traces any write to TMDAT and TMTAG core registers</td>
</tr>
</tbody>
</table>

**NOTE**

For B4 targets, the ownership trace messages will be generated into trace stream only if **Program trace** or **Coverage trace** options available under Trace scenarios are also selected with **Ownership trace** option.

3. In the **Intermediate** tab, you need to configure mode of trace collection and trace control settings.
The following table describes the Intermediate trace and profile options required for hardware profiling.

**Table 3-3. Trace and Profile Intermediate Settings**

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Trace Collection - Mode</strong></td>
<td></td>
</tr>
<tr>
<td>One Buffer</td>
<td>Stops trace collection when trace buffer gets filled.</td>
</tr>
<tr>
<td>Overwrite</td>
<td>Allows trace buffer to work in wrap mode. When trace buffer gets full, trace data continues to get collected overwriting existing data.</td>
</tr>
<tr>
<td>Continuous</td>
<td>The Continuous mode collects trace continuously till you suspend the target application. The continuous trace collection is available only when trace buffer is placed in DDR.</td>
</tr>
<tr>
<td><strong>Trace Collection - Location</strong></td>
<td></td>
</tr>
<tr>
<td>NPC buffer</td>
<td>Saves trace data in NPC internal buffer.</td>
</tr>
<tr>
<td>Gigabit TAP + Trace Probe buffer size (bytes)</td>
<td>Saves trace data in probe buffer.</td>
</tr>
<tr>
<td>DDR buffer</td>
<td>Saves trace data in DDR. Magenta is the bus that transfers trace data from NPC internal buffer to DDR.</td>
</tr>
<tr>
<td>Buffer start address</td>
<td>Specifies the start address of the DDR where the trace data is saved.</td>
</tr>
<tr>
<td>Buffer size</td>
<td>Specifies the DDR memory size of the region used as trace buffer.</td>
</tr>
</tbody>
</table>

*NOTE:* The Trace buffer start address should be aligned to a block boundary of 128 bytes.
### Table 3-3. Trace and Profile Intermediate Settings (continued)

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use settings from LCF file</td>
<td>Uses trace buffer start address and size from linker files. File common.13k contains _TRACE_BUFFER_size and _TRACE_BUFFER_start variables that allows you to change the start address and size. File mmu_attr.13k contains _ENABLE_TB variable that allows to enable and disable settings for trace buffer from linker's files.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE:</strong> The Trace buffer start address should be aligned to a block boundary of 128 bytes.</td>
</tr>
<tr>
<td>Trace control settings - Apply Trace Configuration</td>
<td>The trace collection process is started automatically when the debug session is launched.</td>
</tr>
<tr>
<td>Automatically (When debug session starts)</td>
<td><strong>NOTE:</strong> To successfully collect multicore trace, master core needs to be run into execution before all other cores at first resume of the debug session. If master core is core 0, use multicore resume from Debug toolbar. But if master core is not core 0, master core needs to be resumed first and then click multicore resume from Debug toolbar for all the other cores.</td>
</tr>
<tr>
<td>Manually (Using trace buttons from Trace Commander)</td>
<td>This is the default behavior. This option is used when you want to manually control the trace collection. The trace collection process will not be started automatically on debug session launch, but the trace configuration options are applied, when you click on Trace Start/Stop button available in the Trace Commander view.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE:</strong> To successfully collect multicore trace, master core needs to be run into execution before all other cores at first resume after trace is being started from Trace Commander. If master core is core 0, use multicore resume from Debug toolbar. But if master core is not core 0, master core needs to be resumed first and then click multicore resume from Debug toolbar for all the other cores.</td>
</tr>
<tr>
<td>Trace control settings - Upload Trace Configuration Settings</td>
<td>This is the default behavior. This option is used to upload the trace data automatically.</td>
</tr>
</tbody>
</table>

*Table continues on the next page...*
Table 3-3. Trace and Profile Intermediate Settings (continued)

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NOTE:</strong> The automatic upload trace configuration setting option works only when every suspend of the master trace generator/master core is done after all other trace generators/cores have been suspended to ensure no trace generator generates trace in trace buffer while master trace generator is uploading trace. Otherwise, the trace uploaded will be corrupted.</td>
<td></td>
</tr>
<tr>
<td>Manually (Using trace buttons from Trace Commander)</td>
<td>This option is used when you want to manually upload the trace data using Upload Trace button available in the Trace Commander view.</td>
</tr>
</tbody>
</table>

**NOTE**

**Apply Trace Configuration** and **Upload Trace Configuration** settings are disabled during debug session.

**NOTE**

Trace buffer for B4860 is not per core, but it uses one single buffer for all cores.

4. In the Advanced tab, you need to configure triad settings if you have selected **Profiling - advanced** option available under the **Trace scenarios** list in Basic tab. After selecting **Profiling - advanced** option, the default profiling events are mapped on **Triad A - L1 Icache Access Sorting** and **Triad B - Program L1-> L2 Cacheable Access Sorting**.
The profiling sampling events can be associated to triad A or triad B or both of them.

For example, if you select **Core Events - branch prediction** : **Branch Prediction 1: High level** events in Triad A and for sampling **Instruction Fetching** : **Program access holds** in Triad B the following trace is displayed.

5. In the **Miscellaneous** tab, specify the path for storing the trace and profile results. Click **Browse** to select the path for the **Output Folder**.

### 3.4 Collecting Trace

This section explains the trace collection methods for hardware and simulator based projects.

In this topic:

- Using Simulator
- Using Hardware
3.4.1 Using Simulator

This section explains the trace collection methods for simulator projects.

To start the trace collection:

In the Debug window, click **Debug** to launch the project. The application should halt at the beginning of main in the **Debug** perspective. If you want to modify the trace configuration, click **Configure Trace** button from the **Trace commander** view. You can reconfigure the trace during the debug session. You can apply the new settings, all the data collected before will be deleted and when resuming the session, new trace will be collected with the new configuration.

![Figure 3-23. Debug Perspective - Simulator Screen](Image)

If you have selected **Manual** option in the **Intermediate** page of **Trace and Profile** tab, you should first click **Start Trace Collection** button available in the **Trace commander** view and then **Resume** button to resume the execution and begin measurement. Let the application run for several seconds before performing the next step. During the debug
session, the cycle counter value is enabled and non-zero. When the measurement is stopped, you can view the value of the cycle counter on the status bar as shown below. This value changes after every second when the application is running.

![Figure 3-24. Cycle Counter](image)

If you want to stop the execution of the application while it is running, click Suspend. When you suspend the application, the cycle counter also stops. When the application has executed completely, click Terminate to stop the measurement. When the debug session is terminated, the value of cycle counter becomes zero and disabled.

### 3.4.2 Using Hardware

This section explains the trace collection methods for hardware projects.

To start the trace collection, in the Debug window, click Debug to launch the project.

![Figure 3-25. Debug Perspective - Hardware Screen](image)

For more information on debugging and collecting trace data, refer to the Using Simulator topic.
3.5 Viewing Data

You can view various types of data collected on an application using the Software Analysis view.

On resuming the application, the measurement starts and the trace, timeline, critical code, performance, and call tree data is generated.

Profiling data is not available during debug session, it is available only when you terminate the application. The hyperlinks of profiling results of Software Analysis view are disabled during debug session. Also, if trace collection is stopped before debug session gets terminated, the hyperlinks from Software Analysis view for profiling features will be disabled. Click Refresh button in the Software Analysis view to update the hyperlinks to view the profiling trace data.

To view a data file that is created while running the application on the simulator or hardware:

1. From the menu bar, select Window > Show View > Other > Software Analysis > Software Analysis to open the Software Analysis view.
2. After collecting the data, the data file will be listed in the Software Analysis view as shown in the figure.
   In the Software Analysis view, you can Copy Line or Cell and also Delete Results. These options are available in the context menu of this view.

   **NOTE**
   The Software Analysis view will list the results from the output path in correspondence with the platform configuration selected for that respective project.

![Software Analysis View Screen](image)

3. Expand the datafile to view the trace and profiling data.
The data can be viewed either generated by simulator or hardware.

- Viewing Data Generated by Simulator
- Viewing Data Generated by Hardware

### 3.5.1 Viewing Data Generated by Simulator

This section describes how to view trace data for a simulator project in Software Analysis View.

The following data can be viewed:

- Viewing Trace Data
- Viewing Timeline Data
- Viewing Critical Code Data
- Viewing Performance Data
- Viewing Call Tree Data
- Viewing Counterpoints

#### 3.5.1.1 Viewing Trace Data

Click the **Trace** link to view the trace data generated in Software Analysis view.

The following figure shows the trace data collected for a simulator project.
The below table explains various fields of the trace data.

**Table 3-4. Trace Data**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>Displays the order number of the instructions.</td>
</tr>
<tr>
<td>Event Source</td>
<td>Displays program trace messages from PACC simulator.</td>
</tr>
<tr>
<td>Description</td>
<td>Displays detailed information about the trace line.</td>
</tr>
<tr>
<td>Source</td>
<td>Displays the source function of the trace line if it is a call or a branch.</td>
</tr>
<tr>
<td>Target</td>
<td>Displays the target function of the trace line if it is a call or a branch.</td>
</tr>
<tr>
<td>Type</td>
<td>Displays the type of the trace line, which can either be a linear instruction, a branch, a call, or a custom message.</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Displays the absolute clock cycles that the instruction takes to execute.</td>
</tr>
<tr>
<td>Stalls</td>
<td>Stalls shows the number of stalls that occurred on a PC and measured in cycles.</td>
</tr>
<tr>
<td>ChoF Stalls</td>
<td>Displays the number of bubble cycles incurred by change of program flow for each instruction.</td>
</tr>
<tr>
<td>Interlock Stalls</td>
<td>Displays the number of bubble cycles incurred by core resource interlocks for each instruction.</td>
</tr>
<tr>
<td>Memory Stalls</td>
<td>Displays the number of bubble cycles incurred by data bus hold for each instruction.</td>
</tr>
<tr>
<td>Program Stalls</td>
<td>Displays the number of bubble cycles incurred by program starvation for each instruction.</td>
</tr>
<tr>
<td>Rewind Stalls</td>
<td>Displays the number of bubble cycles incurred by a rewind on the core buses for each instruction.</td>
</tr>
</tbody>
</table>

*Table continues on the next page...*
Table 3-4. Trace Data (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BTB events</td>
<td>The decoded COF is BTBable.</td>
</tr>
<tr>
<td>COF Taken</td>
<td>The COF is taken; changes the program flow.</td>
</tr>
<tr>
<td>False BTB Hit</td>
<td>Indicates False hit.</td>
</tr>
<tr>
<td>BTB prediction</td>
<td>BTB prediction shows the status of the BTB prediction and it can be any of the following:</td>
</tr>
<tr>
<td></td>
<td>• COF predicted correctly: Set of the COF instruction did not cause a pipe flush.</td>
</tr>
<tr>
<td></td>
<td>• COF in BTB: BTBable COF not in the BTB and miss-predicted. It can be taken or not.</td>
</tr>
<tr>
<td></td>
<td>• COF wrongly predicted: BTBable COF, predicted as taken in the BTB, resolved as not taken (wrongly predicted).</td>
</tr>
<tr>
<td></td>
<td>• Buffered sequential loop wrongly predicted: Buffered LPEND incorrectly predicted (Non-BTBable LPEND.SQ with LC&gt;1 and no internal match).</td>
</tr>
<tr>
<td></td>
<td>• BTBable loop wrongly predicted: Either BTB hit on LPEND in last iteration or LPEND taken without BTB hit.</td>
</tr>
</tbody>
</table>

You can also also count the events by the profiling counters. The events are organized into groups of 3, each counted by either Triad A or B. In many cases, two events groups were defined for concurrent profiling, as they belong to the same event category.

The following table summarizes the different counting groups.

Table 3-5. Summary of event groups for profiling

<table>
<thead>
<tr>
<th>Event Category</th>
<th>Event Group</th>
<th>Triad</th>
<th>Counter 0</th>
<th>Counter 1</th>
<th>Counter 2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core events - interrupts</td>
<td>Exceptions 1</td>
<td>A</td>
<td>Trap instructions</td>
<td>Critical interrupts</td>
<td>Non-critical interrupts</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Exceptions 2</td>
<td>B</td>
<td>MMU exceptions</td>
<td>Debug exceptions</td>
<td>Other exceptions</td>
<td>-</td>
</tr>
<tr>
<td>Core events - branch prediction</td>
<td>Branch prediction 1: High level</td>
<td>A</td>
<td>Total VLES</td>
<td>COF VLES (w/o hardware loops)</td>
<td>COF correctly predicted (w/o hardware loops)</td>
<td>Includes both taken and not-taken COFs.</td>
</tr>
<tr>
<td></td>
<td>Branch prediction 2: BTB</td>
<td>B</td>
<td>All BTB-able COF</td>
<td>BTB-able COFs, wrongly predicted, not in the BTB</td>
<td>BTB-able COFs, wrongly predicted, in the BTB</td>
<td>Includes both taken and not-taken COFs.</td>
</tr>
<tr>
<td></td>
<td>Hardware loop prediction</td>
<td>B</td>
<td>Total hardware end-of-loop COF (not including CONT and BREAK)</td>
<td>Incorrectly predicted buffered end-of-loop COF</td>
<td>Other incorrectly predicted end-of-loop COF (not in the BTB, in the BTB wrongly predicted, loop re-learn)</td>
<td>Buffered loops: those sequential loops which fit in the buffer</td>
</tr>
</tbody>
</table>

Table continues on the next page...
### Table 3-5. Summary of event groups for profiling (continued)

<table>
<thead>
<tr>
<th>Event Category</th>
<th>Event Group</th>
<th>Triad</th>
<th>Counter 0</th>
<th>Counter 1</th>
<th>Counter 2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core cycles</td>
<td>Cycle high level</td>
<td>A</td>
<td>Application cycles</td>
<td>-</td>
<td>Bubbles</td>
<td>App.cycles: not in debug and low power modes</td>
</tr>
<tr>
<td>Stall cycles 1</td>
<td>A</td>
<td>No bubbles (once per VLES)</td>
<td>COF (pipe flush) &amp; dispatch starvation cycles</td>
<td>interlocks (RSU) w/o holds</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Stall cycles 2</td>
<td>B</td>
<td>Data memory holds and freezes</td>
<td>&amp;program cache holds</td>
<td>Rewind cycles (prog &amp; data)</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Fetch &amp; Dispatch stall cycles</td>
<td>B</td>
<td>Pipe flush cycles: mispredicted COF, interdicts, exit from idle states</td>
<td>Stalls due to COF target spread over two fetch sets</td>
<td>BTB bubbles</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Rewind events</td>
<td>B</td>
<td>Data rewind events</td>
<td>Program rewind events</td>
<td>DTU rewind events</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Debug events</td>
<td>Debug Events 1</td>
<td>A</td>
<td>IEU Q0</td>
<td>IEU Q1</td>
<td>Reload counter 0 RZ</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Debug Events 2</td>
<td>B</td>
<td>IEU Q2</td>
<td>IEU Q3</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Watchpoints 1</td>
<td>A</td>
<td>-</td>
<td>PCDA0</td>
<td>PCDA2</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Watchpoints 2</td>
<td>B</td>
<td>-</td>
<td>PCDA4</td>
<td>PCDA6</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>DEBUGEV 1</td>
<td>A</td>
<td>DEBUGEV.0</td>
<td>DEBUGEV.2</td>
<td>DEBUGEV.3</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>DEBUGEV 2</td>
<td>B</td>
<td>DEBUGEV.4</td>
<td>DEBUGEV.6</td>
<td>DEBUGEV.7</td>
<td>-</td>
</tr>
<tr>
<td>Trace performance</td>
<td>Trace messages</td>
<td>A</td>
<td>Total trace messages</td>
<td>Error messages</td>
<td>Messages lost</td>
<td>-</td>
</tr>
<tr>
<td>Trace cycle overhead</td>
<td>B</td>
<td>Dual trace beats leaving the queue</td>
<td>Freeze cycles due to main queue (if not frozen due to the input queues)</td>
<td>Freeze cycles due to input queues</td>
<td>Each increment of B0 means two beats sent in parallel to the NPC</td>
<td></td>
</tr>
<tr>
<td>Instruction fetching</td>
<td>L1 Icache access sorting</td>
<td>A</td>
<td>Program access L1 hits</td>
<td>Program access L1 pre-fetch hits</td>
<td>Program access L1 miss</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Program L1-&gt;L2 cacheable access sorting</td>
<td>B</td>
<td>L2 Program access hits (shared/exclusive)</td>
<td>L2 Program access hits (modified)</td>
<td>L2 Program access miss</td>
<td>Only cacheable accesses, including CME accesses</td>
</tr>
<tr>
<td></td>
<td>L1 Program pre-fetch accesses</td>
<td>A</td>
<td>CME/granular PF accesses - L1 hit (including pre-fetch hit)</td>
<td>-</td>
<td>CME granular - L1 miss</td>
<td>Only program CME and cache granular accesses. ctr1 does not count</td>
</tr>
<tr>
<td></td>
<td>Program access holds</td>
<td>B</td>
<td>Program cacheable hold + contention cycles</td>
<td>Program non-cacheable holds + miscellaneous</td>
<td>Program rewind events</td>
<td>The core may continue to execute during program holds</td>
</tr>
</tbody>
</table>

*Table continues on the next page...*
<table>
<thead>
<tr>
<th>Event Category</th>
<th>Event Group</th>
<th>Triad</th>
<th>Counter 0</th>
<th>Counter 1</th>
<th>Counter 2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data channel -</td>
<td>Data access holds - HL</td>
<td>B</td>
<td>Data cache holds</td>
<td>Data store freeze</td>
<td>Data rewind events</td>
<td>B0 and B1 events may overlap</td>
</tr>
<tr>
<td>general</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coherency -</td>
<td>L1 data invalidate from L2</td>
<td>A</td>
<td>L1 data invalidate from</td>
<td>CME/granular invalidation accesses</td>
<td>L1 data invalidation from</td>
<td>Only actual invalidation is counted</td>
</tr>
<tr>
<td>invalidation</td>
<td></td>
<td></td>
<td>L2</td>
<td></td>
<td>internal conflicts</td>
<td></td>
</tr>
<tr>
<td>events</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Loads</td>
<td>L1 Dcache load access sorting</td>
<td>A</td>
<td>Data access L1 hits</td>
<td>Data access L1 pre-fetch hits</td>
<td>Data access L1 miss</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data L1-&gt; L2 cacheable load access sorting</td>
<td>B</td>
<td>L2 Data access hits (shared/</td>
<td>L2 Data access hits (modified)</td>
<td>L2 Data access miss</td>
<td>Only cacheable accesses, including CME accessess</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>exclusive)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>L1 Data pre-fetch accesses</td>
<td>A</td>
<td>CME/granular PF accesses -</td>
<td>CME/granular cache pre-fetch accesses</td>
<td>CME granular - L1 miss</td>
<td>Only data CME and cache granular accesses</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>L1 hit (including pre-fetch hit)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data load holds 1</td>
<td>A</td>
<td>Data load cacheable hold</td>
<td>Data loads non-cacheable hold cycles</td>
<td>Data cache contention</td>
<td>Holds for parallel accesses counted once</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cycles</td>
<td></td>
<td>cycles</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data load holds 2</td>
<td>B</td>
<td>Full fetch queue hold</td>
<td>miscellaneous hold cycles (barriers, special commands)</td>
<td>Address queue load after store hazard cycles</td>
<td>Address queue holds may overlap the data cache holds</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>hold cycles</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data load special cases</td>
<td>A</td>
<td>Total memory loads</td>
<td>Non-aligned 4K loads</td>
<td>Load rewind events</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data load access speculation</td>
<td>B</td>
<td>Total core accesses</td>
<td>Total valid accesses</td>
<td>Killed accesses with cache miss</td>
<td>W bus accesses</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>without cache commands and barriers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data stores</td>
<td>Store access sorting</td>
<td>A</td>
<td>Total store accesses</td>
<td>SGB merges</td>
<td>SGB write out hot (DLINK to L1)</td>
<td>No cache commands and barriers, after non-alignm ent split, merges may be 4 at a time</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SGB events</td>
<td>B</td>
<td>SGB freeze cycles</td>
<td>SGB read-modify-write accesses to L2</td>
<td>SGB accepter rewind</td>
<td>For parallel accesses, a freeze is counted once</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>events</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CME</td>
<td>A</td>
<td>CME data commands</td>
<td>CME SkyBlue data commands</td>
<td>Granular data commands</td>
<td>CME command = actual channel allocation. A2 is not implemente d in the CME</td>
</tr>
</tbody>
</table>

*Table continues on the next page...*
<table>
<thead>
<tr>
<th>Event Category</th>
<th>Event Group</th>
<th>Triad</th>
<th>Counter 0</th>
<th>Counter 1</th>
<th>Counter 2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program CME commands</td>
<td>B</td>
<td>CME program commands</td>
<td>CME SkyBlue program commands</td>
<td>Granular program commands</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>CME allocation</td>
<td>A</td>
<td>Total CME allocation requests (program + data)</td>
<td>Total successful allocations (program + data)</td>
<td>Total CME channel substitutions (program + data)</td>
<td>Substitution s = incomplete pre-fetch channel</td>
<td></td>
</tr>
<tr>
<td>Bus load</td>
<td>L1-L2 bus load</td>
<td>A</td>
<td>DLINK read beats</td>
<td>DLINK write beats</td>
<td>ILINK beats</td>
<td>A single load access (A0,A2) can be 2 bus cycles (beats) if cacheable. Cache commands (no-tag access) counted as writes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B</td>
<td>Core access to local subsystem registers in the CCSR space</td>
<td>SkyBlue access from the SoC or adjacent core in the cluster to local subsystem registers in the CCSR space</td>
<td>SkyBlue access from any source to the local subsystem registers in the DCSR space</td>
<td></td>
</tr>
<tr>
<td>SkyBlue bus load</td>
<td></td>
<td>B</td>
<td>Core access to local subsystem registers in the CCSR space</td>
<td>SkyBlue access from the SoC or adjacent core in the cluster to local subsystem registers in the CCSR space</td>
<td>SkyBlue access from any source to the local subsystem registers in the DCSR space</td>
<td></td>
</tr>
<tr>
<td>L2 cache statistics</td>
<td>L2 demand (Elink) accesses - 1</td>
<td>A</td>
<td>Total L2 demand accesses</td>
<td>L2 program accesses</td>
<td>L2 data accesses</td>
<td>Each counter counts up to 4 events per cycle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B</td>
<td>Total L2 hit</td>
<td>L2 instruction miss</td>
<td>L2 data miss</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>A</td>
<td>Total L2 AOUT requests</td>
<td>L2 AOUT requests sent as global (M=1)</td>
<td>L2 AOUT data side requests (including barriers)</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B</td>
<td>Total L2 demand (ELINK) accesses</td>
<td>L2 Snoop requests</td>
<td>Tot. L2 accesses from all sources</td>
<td>Counters 0 and 2 may count up to 4 events per cycle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A</td>
<td>L2 Total stashes</td>
<td>L2 Snoop requests</td>
<td>L2 Stash requests degraded to snoop</td>
<td>KEXT should be counted with either KDIN or KSNP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B(a)</td>
<td>L2 reload requests from CoreNet</td>
<td>L2 Store allocate</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B(b)</td>
<td>L2 Snoop hit</td>
<td>L2 snoops causing MINT</td>
<td>L2 snoops causing SINT</td>
<td></td>
</tr>
</tbody>
</table>
To copy the cell that is currently selected, select the Copy Cell option from the context menu. To copy the complete line of the data source, select the Copy Line option. If you want to resize the row or column, select the Auto Resize Row or Auto Resize Column Width option from the context menu.

The Trace viewer displays two types of trace data:

- Partially decoded trace data
- Fully decoded trace data

Full decoded trace is displayed if prior to opening Trace viewer any of the profiling viewers (Timeline, Critical Code, Performance, Call Tree, Counterpoints) is already open. If none of the profiling viewers has been opened, then Trace viewer will display the partial decoded trace or trace decoded on chunks. Once trace is decoded, Trace viewer will always display full decoded trace.

Partial decoded trace has certain limitations because current chunk of trace to be decoded might not contain all information needed to decode that chunk, for example, chunk might lack PC synchronization or RAS stack information in scenarios when trace is collected in overwrite mode or multicore core. Such limitations happen only on program trace and not on profiling trace.

The following table lists the details of each trace viewer event showing the limitations of partial decoded trace:

<table>
<thead>
<tr>
<th>Trace Events</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Periodic synchronization</td>
<td>After periodic synchronization message no trace dropping should happen.</td>
</tr>
<tr>
<td>Trace dropped - no PC synchronization</td>
<td>Trace has been dropped because no message with PC (program counter) has been analyzed.</td>
</tr>
<tr>
<td>Trace partially recovered - PC synchronization but no RAS stack</td>
<td>Trace has partially recovered due to a message providing PC. Message however, does not ensure a synchronization for RAS stack. Therefore, it is expected that trace to be dropped once destination address for a RTS instruction from RAS stack will be needed.</td>
</tr>
<tr>
<td>Trace dropped - RTS treated as direct COF but RAS stack is empty</td>
<td>Trace has been dropped, because RTS instruction analyzed requires its destination from RAS stack but stack is empty.</td>
</tr>
<tr>
<td>Trace fully recovered - PC and RAS stack synchronization</td>
<td>Trace has fully recovered due to a message that provides both PC and RAS stack synchronized. This kind of message is expected to come once with Periodic synchronization message.</td>
</tr>
<tr>
<td>Trace dropped - certain type of instruction incorrectly traced</td>
<td>Trace has been dropped due to an unexpected error in decoding algorithm.</td>
</tr>
<tr>
<td>Call</td>
<td>It reports a call instruction from a source function to a destination function.</td>
</tr>
</tbody>
</table>

Table continues on the next page...
Table 3-6. Trace viewer events (continued)

<table>
<thead>
<tr>
<th>Trace Events</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>It reports a return instruction from a source function to a destination function.</td>
</tr>
<tr>
<td>Branch</td>
<td>It reports a branch instruction from a source function to a destination function.</td>
</tr>
<tr>
<td>Linear</td>
<td>It reports a linear instruction.</td>
</tr>
<tr>
<td>Debug Status</td>
<td>It reports core state.</td>
</tr>
<tr>
<td>Device Id</td>
<td>It reports device id.</td>
</tr>
<tr>
<td>User defined</td>
<td>It reports any write made by user to TMDAT and TMTAG core registers.</td>
</tr>
<tr>
<td>Ownership</td>
<td>It reports information on current task ID.</td>
</tr>
<tr>
<td>Profiling counter overflow</td>
<td>It reports an overflow of the profiling counters.</td>
</tr>
</tbody>
</table>

The Trace Data viewer provides a color convention in the form of Legend to highlight trace events and source changes. The details on the color convention is listed in the table below:

Table 3-7. Trace Data

<table>
<thead>
<tr>
<th>Color</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>Highlights the error events reported in the trace data.</td>
</tr>
<tr>
<td>Orange</td>
<td>Highlights interrupt jumps reported in both program and profiling trace data.</td>
</tr>
<tr>
<td>Green</td>
<td>Highlights the info events reported in the trace data.</td>
</tr>
<tr>
<td>Purple (Light and Dark)</td>
<td>Highlights the data events reported in the trace data; light purple alternates with dark purple for odd and even lines.</td>
</tr>
<tr>
<td>Blue (Light and Dark) and White (White and Seashell)</td>
<td>Highlights all other trace events. Blues alternate with whites as source changes. Light blue alternates with dark blue for odd and even lines. White alternates with seashell for odd and even lines.</td>
</tr>
</tbody>
</table>

3.5.1.1.1 Exporting Trace Data

In the trace data viewer you can export trace data to a CSV file.

Perform the following steps to export trace data:

1. Click to display the Exporting Trace Data dialog box.
2. Select the **Export Full Trace** option to export complete trace data to a CSV file else select **Export Partial Trace**.
3. Specify the begin and end index of the trace data if you have selected the **Export Partial Trace** option.
4. Click **OK**. The **Export Trace Data to CSV** dialog box appears.
5. Browse to the location where you want to save the trace data.

### 3.5.1.1.2 Configuring Time Unit and Time Format

The **Configure Table** button allows you to **Configure Time Unit** and **Configure Time Format** actions.

- **Configure Time Unit** - Lets you set CPU frequency and convert the clock cycles, displayed in the **Timestamp** column, into real time in milliseconds, microseconds, or nanoseconds. Click this button and select the **Configure Time Unit** option as shown in the figure. To convert the clock cycles into milliseconds, microseconds, or nanoseconds, select the corresponding option.
The **Set CPU Frequency** option allows you to set the CPU frequency needed to convert the clock cycles into real time. Select this option to display the **Set CPU Frequency** dialog box. Set the new CPU frequency according to requirements.

- **Configure Time Format** - Enables the **Timestamp** column of the Trace Data viewer for time formatting. **Radix**: *Hexadecimal or Decimal* and **Value**: *Absolute or Delta* as shown in the figure below. Time radix formatting is available only if time unit is set to **Time in cycles**. The default format of time is displayed in Decimal and Absolute value. Delta is calculated for same source event. The Delta value for a specific timestamp is the difference between current timestamp and previous timestamp with same source event as current one. If Delta value is negative, it will be displayed as zero.

![Configure Time Format](image)

**Figure 3-30. Configure Time Format**

### 3.5.1.2 Viewing Timeline Data

To view the timeline data, click the **Timeline** link.

The following figure shows an example of Timeline data.

![Timeline Data](image)

**Figure 3-31. Timeline Data**
The timeline data displays the functions that are executed in the application and the number of cycles each function takes when the application is run. The **Timeline** viewer shows the function percent of the completion status for each function in the left panel and timeline graph in the right panel with the functions aligned on y-axis and the number of cycles on x-axis. The green-colored bars show the time and cycles that the function takes.

The Timeline viewer have the following buttons:

- **Selection Mode** - Allows you to mark points in the function bars to measure the difference of cycles between those points. To mark a point in the bar:
  - Click **Selection Mode**.
  - Click on the bar where you want to mark the point.
  - A yellow vertical line appears displaying the number of cycles at that point.
  - Right-click on another point in the bar.
  - A red vertical line appears displaying the number of cycles at that point along with the difference of cycles between two marked points as shown in the below figure.

![Figure 3-32. Selection Mode to Measure Difference of Cycles Between Functions](image)

- **Zoom Mode** - Allows you to zoom-in and zoom-out in the timeline graph. Click **Zoom Mode** and then click on the timeline graph to zoom-in. To zoom-out, right-click on the timeline graph. You can also move the mouse wheel up and down to zoom-in and zoom-out.
- **Full View** - Allows you to get back to the original view if you selected the zoom mode.

**NOTE**

The **Selection Mode** is the default mode of the timeline view.
• **Configure Table** - Lets you configure time unit in cycles, milliseconds, microseconds, and nanoseconds. For example, to display time displayed in the timeline graph in milliseconds, select **Configure Table > Configure Time Unit > Milliseconds**. It also allows you to set CPU frequency. For details, refer to the Configuring Time Unit and Time Format topic.

• **Export timeline graphic to file** - Allows you to save the data visible in the timeline view as an image in .JPG or .PNG format.

• **Edit Groups** - Lets you customize the timeline according to your requirements. For example, you can change the default color of the line bars representing the functions to differentiate between them. You can add/remove a function to/from the timeline. To perform these functions, select **Edit Groups**. The **Edit Groups** dialog box appears.

![Edit Groups Dialog Box](figure-3-33.png)

**Figure 3-33. Edit Group Dialog Box**

You can perform the following operations in the **Edit Groups** dialog box:

• Add or Remove Function

• Edit Address Range of Function
3.5.1.2.1 Add or Remove Function

This section explains how to add or remove a function from the Edit Groups dialog box. Right-click on the function name in the Name column, and select Insert Function or press Ctrl+F to add a function. Select Delete Selected from the context menu to delete the function from the graph. You can disable a function from the graph by clearing the corresponding checkbox in the Name column. Check it again to include it in the graph.

3.5.1.2.2 Edit Address Range of Function

This section explains how to edit the address range of a function from the Edit Groups dialog box.

Perform the following steps:

1. Select the function of which you want to change the address range.
2. Double-click on the cell of the Addresses column of the selected function.
   The cell becomes editable.
3. Type an address range for the group or function in the cell.

   NOTE
   You can specify multiple address ranges to a function. The multiple address ranges are separated by a comma.

3.5.1.2.3 Change Color

This section explains how to change the color of a function in the timeline graph.

The color appears as a horizontal bar in the graph. Click on the Color column of the corresponding function, and select the color of your choice from the Color window that appears.

3.5.1.2.4 Add or Remove Group
This section explains how to add or remove a group from **Edit Groups** dialog box.

A group is a range of addresses. In case, you want to view trace of a part of a function only, for example, for loop, you can find the addresses of the loop and create a group for those addresses. Perform the following steps to add a group:

1. Right-click on the row, in the **Edit Groups** dialog box, where you want to insert a group, and select **Insert Group** from the context menu. Alternatively, press the Ctrl + G key.

   A row is added to the table with `new` as function name.

   ![Figure 3-34. Adding Group Dialog Box](image)

2. Double-click on the `new` group cell.

   The cell becomes editable.

3. Type a name for the group, for example, **MyGroup**.

4. Double-click on the cell of the corresponding **Addresses** column, and edit the address range according to requirements, for example, **0x1fff0300-0x1fff035f**.

5. Change the color of the group.

---

*CodeWarrior Development Studio for StarCore SC3900FP DSP Architectures Tracing and Analysis Tools User Guide, Rev. 10.9.0, 11/2015*

Freescale Semiconductor, Inc.
To delete a group, select it, right-click on the Edit Groups dialog box, and select the Delete Selected option from the context menu. You can also remove a group from the graph by clearing the corresponding checkbox in the Name column. Check it again to include it in the graph.

**3.5.1.2.5 Merge Groups or Functions**

This section explains how to merge the groups or functions available.

Merging is useful in case there are many functions and you do not want to view the trace of each and every function. You cannot undo this operation, that is you cannot separate the merged functions or groups. To view the original trace data, reopen the Trace Data viewer. Perform the following steps to merge the group or function:

1. In the Edit Groups dialog box, select the function or group to be merged.
2. Drag and drop it in the function or group with which you want it to get merged with.
3. Both the functions or groups merge into a single function or group that covers both address ranges, where the function, `main` is merged with the group, `MyGroup`.

![Edit Groups](image)

**Figure 3-36. Merging Functions or Groups Screen**

4. Click **OK**.

### 3.5.1.3 Viewing Critical Code Data

To view the critical code data, click the **Critical Code** link.

The following diagram shows an example of Critical Code data.
The critical code data displays the summarized data of a function in a tabular form. The columns are movable; you can drag and drop the columns to move them according to your requirements.

The below table explains the various fields of Critical Code tab.

**Table 3-8. Critical Code Data**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>Displays the start address of the function.</td>
</tr>
<tr>
<td>Function</td>
<td>Displays the name of the function that has executed.</td>
</tr>
<tr>
<td>Coverage %</td>
<td>Displays the percentage of number of assembly instructions executed from the total number of assembly instructions in a function.</td>
</tr>
<tr>
<td>ASM Decision Coverage %</td>
<td>Displays the decision coverage computed for direct and indirect conditional branches. It is the mean value of the individual decision coverages. So if a function has two conditional instructions, one with 100% and another with 50% decision coverage, the decision coverage would be ((100 + 50) / 2 = 75%). It is calculated only for assembly instructions and not for C source code.</td>
</tr>
<tr>
<td>ASM Count</td>
<td>Displays the number of lines executed in the function.</td>
</tr>
<tr>
<td>Time (Microsecond)</td>
<td>Displays the execution time of each function.</td>
</tr>
<tr>
<td>Size</td>
<td>Displays the number of bytes required by each function.</td>
</tr>
<tr>
<td>Chof Stalls</td>
<td>Displays the number of bubble cycles incurred by change of program flow for each function.</td>
</tr>
<tr>
<td>Interlock Stalls</td>
<td>Displays the number of bubble cycles incurred by core resource interlocks for each function.</td>
</tr>
<tr>
<td>Memory Stalls</td>
<td>Displays the number of bubble cycles incurred by data bus hold for each function.</td>
</tr>
</tbody>
</table>

*Table continues on the next page...*
Table 3-8. Critical Code Data (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Stalls</td>
<td>Displays the number of bubble cycles incurred by program starvation for each function.</td>
</tr>
<tr>
<td>Rewind Stalls</td>
<td>Displays the number of bubble cycles incurred by a rewind on the core buses for each function.</td>
</tr>
</tbody>
</table>

The **Critical Code** tab displays data into two views; the top view displays the summary of the functions, and the bottom view displays the statistics for all the instructions executed in a particular function. Click on a hyperlinked function in the top view of the **Critical Code** viewer to view the corresponding statistics for the instructions executed in that function. For example, the statistics of the `main()` function is shown below.

![Figure 3-38. Statistics of Critical Code Data](image)

The below table describes the fields of the statistics of the critical code data.

Table 3-9. Description of Statistics of Critical Code Data

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line/Address</td>
<td>Displays either the line number for each instruction in the source code or the address for the assembly code.</td>
</tr>
<tr>
<td>Instruction</td>
<td>Displays all the instructions executed in the selected function.</td>
</tr>
<tr>
<td>Coverage %</td>
<td>For C source lines, displays the percentage of number of assembly instructions executed from the total number of assembly instructions corresponding to the source line. For assembly source lines, it shows if the instructions were executed or not.</td>
</tr>
</tbody>
</table>
Table 3-9. Description of Statistics of Critical Code Data (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASM Decision Coverage</td>
<td>Displays the decision coverage computed for direct and indirect conditional branches. It is the mean value of the individual decision coverages. So if a function has two conditional instructions, one with 100% and another with 50% decision coverage, the decision coverage would be ((100 + 50) / 2 = 75%). It is calculated only for assembly instructions and not for C source code.</td>
</tr>
<tr>
<td>ASM Count</td>
<td>Displays the number of times each instruction is executed.</td>
</tr>
<tr>
<td>Time (CPU Cycles)</td>
<td>Displays the total time taken by each instruction in the function.</td>
</tr>
<tr>
<td>Chof Stalls</td>
<td>Displays the number of bubble cycles incurred by change of program flow for each function.</td>
</tr>
<tr>
<td>Interlock Stalls</td>
<td>Displays the number of bubble cycles incurred by core resource interlocks for each instruction in the function.</td>
</tr>
<tr>
<td>Memory Stalls</td>
<td>Displays the number of bubble cycles incurred by data bus hold for each function.</td>
</tr>
<tr>
<td>Program Stalls</td>
<td>Displays the number of bubble cycles incurred by program starvation for each function.</td>
</tr>
<tr>
<td>Rewind Stalls</td>
<td>Displays the number of bubble cycles incurred by a rewind on the core buses for each function.</td>
</tr>
</tbody>
</table>

**NOTE**

In the Critical Code viewer, all functions in all files associated with the project are displayed irrespective of coverage percentage. However, the 0% coverage functions do not appear in the Performance and Call Tree viewers because these functions are not considered to be computed and are not a part of caller-called pair.

Click on the column header to sort the critical code data by that column. However, you can only sort the critical code data available on the top view. The icons available in the statistics view of the Critical Code tab allow you to perform the following actions:

- **Previous function** - Lets you navigate to the previous function you selected in the File/Function column.
- **Next function** - Lets you navigate to the next function in the File/Function column.
- **Export** - Lets you export the critical code data of both top and bottom views in a CSV or HTML file.
- **Configure Table** - Lets you show and hide column(s) of the critical code data. Click and select the **Configure the table above option** to show/hide columns of the top view or the **Configure the table below option** to show/hide columns of the bottom view. The **Drag and drop to order columns** dialog box appears in which you can check/uncHECK the checkboxes corresponding to the available columns to show/hide.
them in the Critical Code viewer. The option also allows you to set CPU frequency and set time in cycles, milliseconds, microseconds, and nanoseconds.

- **Collapse/Expand all files** - Lets you collapse/expand all files in the Summary Table.
- **Filter files** - Lets you filter the desired files to display in the Summary Table. Click this button and select the desired files from the Choose files to be listed dialog.
- **Switch to executable source line statistics/Switch to ASM instructions statistics** - Lets you display the executable source line statistics/ASM instructions statistics data format in the Summary Table.
- **Graphics** - Lets you display the colored bars for the columns in the bottom view of the critical code data. For example, click and select the Assembly > ASM Count, option to display colored bars in the ASM Count. The colors in these columns differentiate source code with the assembly code.
- **Show code** - Lets you display the assembly or mixed code in the statistics of the critical code data. Click and select the Assembly option to display the assembly code. Similarly, select the Source or the Mixed option to display the source or mixed code in the statistical details.

### 3.5.1.3.1 Export Critical Code Data

You can export critical code statistical data into CSV or HTML file format.

To export the data, follow the given procedure:

1. Click the export button available on the Critical Code view.

   The drop-down menu shows Export to CSV or Export to HTML option.

2. Select any one of the following option to export the statistical data in the respective format:
   - **Export to CSV**: Select the Export summary table statistics option to export the details of the top view or the Export Details table statistics option to export the details of the bottom view respectively.
   - **Export to HTML**: Select Export as source option to export the data in function details table that contains source code lines, Export as asm option to export the data in function details table that contains address and assembly code of the function, and Export as mixed option to export the data in function details table that contains both addresses and line numbers of the instructions with their corresponding source and assembly code.

   The Export to dialog box appears.
3. Specify the File name and browse the location where you want to save the data.
4. Click Save.

The exported data will be saved in the respective format (.csv or .html).

The data exported on the html page is structured in the given format:

- **Files**: Lists all the analyzed files
- **File Analysis Content**: Provides details on Function Coverage table, Each function detail (including Details table, Code coverage metrics table, and ASM decision metrics table), Summary table, and ASM decision coverage table of the selected file.

**NOTE**

While reading the data from the html page, the content of Function Details table suggests what option was selected from Export to HTML menu (Export as source, asm or mixed). For example, if the first column name of the Function Details table is Line, the option selected is Export as source. Similarly, if the first column name is Address, then the option selected is Export as asm, and if the first...
column name is Line/Address then the option selected is Export as mixed.

The screens given below shows the data structured in different tables:

**Function coverage for sc3900_main.c**

<table>
<thead>
<tr>
<th>Function</th>
<th>Covered %</th>
</tr>
</thead>
<tbody>
<tr>
<td>func2</td>
<td>100 %</td>
</tr>
<tr>
<td>vector_dot_product_16x16</td>
<td>100 %</td>
</tr>
<tr>
<td>main</td>
<td>62 %</td>
</tr>
</tbody>
</table>

**Figure 3-40. Function Coverage Table**

**Source table for func2**

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>58</td>
<td>`{</td>
<td>covered</td>
</tr>
<tr>
<td>59</td>
<td>#pragma noinline</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>printf(&quot;Hello private text\n&quot;);</td>
<td>covered</td>
</tr>
<tr>
<td>62</td>
<td>return 0;</td>
<td>covered</td>
</tr>
<tr>
<td>63</td>
<td>}</td>
<td>covered</td>
</tr>
</tbody>
</table>

**Code coverage metrics for func2**

<table>
<thead>
<tr>
<th>Total ASM instructions</th>
<th>Covered ASM instructions</th>
<th>Not covered ASM instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

**ASM decision metrics table for func2**

<table>
<thead>
<tr>
<th>Total ASM decisions</th>
<th>Taken ASM decisions</th>
<th>Not taken ASM decisions</th>
<th>Both taken ASM decisions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Figure 3-41. Function Details Table**

**Summary table for sc3900_main.c**

<table>
<thead>
<tr>
<th>Function</th>
<th>Total ASM instructions</th>
<th>Covered ASM instructions</th>
<th>Coverage %</th>
</tr>
</thead>
<tbody>
<tr>
<td>func2</td>
<td>6</td>
<td>6</td>
<td>100 %</td>
</tr>
<tr>
<td>vector_dot_product_16x16</td>
<td>284</td>
<td>284</td>
<td>100 %</td>
</tr>
<tr>
<td>main</td>
<td>50</td>
<td>31</td>
<td>62 %</td>
</tr>
<tr>
<td>Whole program</td>
<td>340</td>
<td>321</td>
<td>94.41 %</td>
</tr>
</tbody>
</table>

**Figure 3-42. Summary Table**
3.5.1.4 Viewing Performance Data

To view performance data, click on the Performance hyperlink.

The Performance viewer appears as shown below.

The Performance viewer is divided into two views:

- The top view presents function performance data in the Summary table. It displays the count and invocation information for each function that executes during the measurement, enabling you to compare the relative data for various portions of your target program. The information in the Summary table can be sorted by column in ascending or descending order. Click the column header to sort the corresponding data.

<table>
<thead>
<tr>
<th>Function</th>
<th>Total ASM decisions</th>
<th>ASM Decision Coverage %</th>
</tr>
</thead>
<tbody>
<tr>
<td>func2</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>vector_dot_product_16x16</td>
<td>4</td>
<td>62.5%</td>
</tr>
<tr>
<td>main</td>
<td>2</td>
<td>50%</td>
</tr>
<tr>
<td>Whole program</td>
<td>6</td>
<td>58.33%</td>
</tr>
</tbody>
</table>

Figure 3-43. ASM Decision Coverage Table

Figure 3-44. Performance Viewer
The following table explains the fields of the Summary table.

### Table 3-10. Field description of Summary table

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function Name</td>
<td>Name of the function that has executed.</td>
</tr>
<tr>
<td>Num Calls</td>
<td>Number of times the function has executed.</td>
</tr>
<tr>
<td>Inclusive</td>
<td>Cumulative metric count during execution time spent from function entry to exit.</td>
</tr>
<tr>
<td>Min Inclusive</td>
<td>Minimum metric count during execution time spent from function entry to exit.</td>
</tr>
<tr>
<td>Max Inclusive</td>
<td>Maximum metric count during execution time spent from function entry to exit.</td>
</tr>
<tr>
<td>Avg Inclusive</td>
<td>Average metric count during execution time spent from function entry to exit.</td>
</tr>
<tr>
<td>Percent Inclusive</td>
<td>Percentage of total metric count spent from function entry to exit.</td>
</tr>
<tr>
<td>Exclusive</td>
<td>Cumulative metric count during execution time spent within function.</td>
</tr>
<tr>
<td>Min Exclusive</td>
<td>Minimum metric count during execution time spent within function.</td>
</tr>
<tr>
<td>Max Exclusive</td>
<td>Maximum metric count during execution time spent within function.</td>
</tr>
<tr>
<td>Avg Exclusive</td>
<td>Average metric count during execution time spent within function.</td>
</tr>
<tr>
<td>Percent Exclusive</td>
<td>Percentage of total metric count spent within function.</td>
</tr>
<tr>
<td>Percent Total Calls</td>
<td>Percentage of the calls to the function compared to the total calls.</td>
</tr>
<tr>
<td>Code Size</td>
<td>Number of bytes required by each function.</td>
</tr>
</tbody>
</table>

- The bottom view or the **Details** table presents call pair data for the function selected in the **Summary** table. It displays call pair relationships for the selected function, that is which function called the another function. Each function pair consists of a caller and a callee. The percent caller and percent callee data is also displayed graphically. The functions are represented in different colors in the pie chart, you can move the mouse cursor over the color to see the corresponding function.

The below table describes the fields of the **Details** table. You cannot sort the columns of this table.

### Table 3-11. Field description of Details table

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caller</td>
<td>Name of the calling function.</td>
</tr>
</tbody>
</table>
Table 3-11. Field description of Details table (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Callee</td>
<td>Name of the function that is called by the calling function.</td>
</tr>
<tr>
<td>Num Calls</td>
<td>Number of times the caller called the callee.</td>
</tr>
<tr>
<td>Inclusive</td>
<td>Cumulative metric count during execution time spent from function entry to exit.</td>
</tr>
<tr>
<td>Min Inclusive</td>
<td>Minimum metric count during execution time spent from function entry to exit.</td>
</tr>
<tr>
<td>Max Inclusive</td>
<td>Maximum metric count during execution time spent from function entry to exit.</td>
</tr>
<tr>
<td>Avg Inclusive</td>
<td>Average metric count during execution time spent from function entry to exit.</td>
</tr>
<tr>
<td>Percent Callee</td>
<td>Percent of total metric count during the time the selected function is the caller of a specific callee.</td>
</tr>
<tr>
<td>Percent Caller</td>
<td>Percent of total metric count during the time the selected function is the callee of a specific caller.</td>
</tr>
<tr>
<td>Call Site</td>
<td>Address from where the function was called.</td>
</tr>
</tbody>
</table>

The table below lists the buttons available in the Summary table of the Performance viewer.

Table 3-12. Buttons Available in Summary Table of Performance Viewer

<table>
<thead>
<tr>
<th>Name</th>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous Function</td>
<td><img src="image" alt="Button" /></td>
<td>Lets you view the details of the previous function that was selected in the bottom view before the currently selected function. Click it to view the details of the previous function.</td>
</tr>
<tr>
<td>Next Function</td>
<td><img src="image" alt="Button" /></td>
<td>Lets you view the details of the next function that was selected in the bottom view. NOTE: The Previous and Next buttons are contextual and go to previous/next function according to the history of selections. So if you select a single line in the view, these buttons will be disabled because there is no history.</td>
</tr>
<tr>
<td>Export</td>
<td><img src="image" alt="Button" /></td>
<td>Lets you export the performance data of both top and bottom views in a CSV file. Click the button and select the Export the statistics above option to export the details of the top view or the Export the statistics below option to export the details of the bottom view respectively.</td>
</tr>
<tr>
<td>Configure Table</td>
<td><img src="image" alt="Button" /></td>
<td>Lets you show and hide column(s) of the performance data. Click the button and select the Configure the table above option to show/hide columns of the top view or the Configure the table below option to show/hide columns of the bottom view. The Drag and drop to order columns dialog box appears in which you can check/clear the checkboxes corresponding to the available columns to show/hide them in the Performance viewer. The Configuring Time Unit option allows you to set CPU frequency and set time in cycles, milliseconds, microseconds, and nanoseconds.</td>
</tr>
</tbody>
</table>

Table continues on the next page...
Table 3-12. Buttons Available in Summary Table of Performance Viewer (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exclude Symbols</td>
<td></td>
<td>Lets you exclude statistics libraries or symbols from the performance data. Click this button to select/specify a function or library you want to exclude from the performance data. After excluding the selected function, library, or symbol, the statistics will be recomputed and reloaded in the Performance data viewer. For details, see Exclude Symbols from Statistics.</td>
</tr>
<tr>
<td>Choose view</td>
<td></td>
<td>Lets you to switch between the data available in the Metrics view and Statistics view of the summary table. Metrics view displays all available performance statistics details for a specific metric; for example, a Metrics view for Cycles metric displays all available performance statistics details for metric Cycles. Statistics view displays all performance metrics details for a specific statistic; for example, a Statistics view for Inclusive statistic displays all available performance metrics details for Inclusive statistic. NOTE: Performance details table is available only for metrics view. The data in Details table is refreshed when a user selects a new metric. NOTE: For Statistics view Configure Columns for Details Table and Export Details table statistics options are disabled.</td>
</tr>
<tr>
<td>Choose metric/statistic</td>
<td></td>
<td>Lets you load and list the metrics name (Cycles) or statistics name (Inclusive) in the summary table of the Performance data. The data displayed in the Summary table will correspond to the selection made in the Choose view and Choose metric/statistic fields.</td>
</tr>
</tbody>
</table>

By default, the data collected in the Performance data viewer is based on Cycle metric and is displayed in the Metrics view. The below figure shows the Statistics view of the Performance data viewer.

Figure 3-45. Performance Viewer - Statistics View
3.5.1.5 Viewing Call Tree Data

To view the call tree data, click the **Call Tree** link.

The **Call Tree** viewer appears as shown below.

![Call Tree Viewer](image)

**Figure 3-46. Call Tree Viewer**

In the **Call Tree** viewer, **START** is the root of the tree. You can click on "+" to expand the tree and "-" to collapse the tree. It shows the biggest depth for stack utilization in **Call Tree** and the functions on this call path are displayed in green color. The Call Tree nodes are synchronized with the source code. You can double-click on the node to view the source code. The columns are movable; you can move the columns to the left or right of another column depending on your requirements by dragging and dropping.

The following table describe the fields of **Call Tree** data.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function Name</td>
<td>Name of function that has executed.</td>
</tr>
<tr>
<td>Num Calls</td>
<td>Number of times function has executed.</td>
</tr>
<tr>
<td>% Total calls of Parent</td>
<td>Percent of number of function calls from total number of calls in the application.</td>
</tr>
<tr>
<td>% Total times it was called</td>
<td>Percent of number of times a function was called.</td>
</tr>
<tr>
<td>Inclusive Time</td>
<td>Cumulative count during execution time spent from function entry to exit.</td>
</tr>
</tbody>
</table>

You can set **CPU frequency** and **Inclusive Time** displayed in the **Call Tree** viewer in cycles, milliseconds, microseconds, and nanoseconds. Click **Configure Table** to configure time unit and set CPU frequency. For details, refer to the Configuring Time Unit and Time Format topic.
The table below lists the toolbar buttons available in the Call Tree view.

<table>
<thead>
<tr>
<th>Name</th>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configure Table</td>
<td><img src="image" alt="Configure Table" /></td>
<td>Lets you set CPU frequency and set time in cycles, milliseconds, microseconds, and nanoseconds using the Configuring Time Unit option.</td>
</tr>
<tr>
<td>Export to dot</td>
<td><img src="image" alt="Export to dot" /></td>
<td>Lets you export the call tree data to .dot file. Click this button to open the Export to dot dialog and save the data as .dot file in the desired location.</td>
</tr>
<tr>
<td>Exclude Symbols</td>
<td><img src="image" alt="Exclude Symbols" /></td>
<td>Lets you exclude statistics libraries or symbols from the performance data. Click this button to select/specify a function or library you want to exclude from the call tree data. After excluding the selected function, library, or symbol, the statistics will be recomputed and reloaded in the Call Tree data viewer. For details, see Exclude Symbols from Statistics.</td>
</tr>
</tbody>
</table>

### 3.5.1.6 Viewing Counterpoints

To view Counterpoints information, click on the Counterpoints hyperlink.

Counterpoints allow you to count events between two points of the executed code. The results are computed based on the collected trace data; the counterpoints are searched in trace by their addresses. For information on counterpoints, refer to the Viewing Counterpoints in Analysispoints topic.

### 3.5.2 Viewing Data Generated by Hardware

This section describes how to view trace data for a hardware project.

In the Basic subtab of Trace and Profile tab of Debug window, the following viewers are available in the Software Analysis view:

- If you have selected either Program Trace or Coverage scenarios, only Trace and Critical Code viewers are displayed.
- If you have selected any profiling trace scenario, Trace, Performance, Timeline, Critical Code, and Call Tree viewers are displayed.
- If you have selected User defined events or/and Ownership (used with None scenario), only Trace viewer is displayed.
- Log and Counterpoints viewers are displayed with any combination of trace scenarios.
Also, a note has been added in Basic tab of Trace and Profile that informs you about the kind of trace that will be collected with the chosen configuration. In case Profiling-advanced scenario is selected, you will have to customize the configuration in Advanced tab. In case User defined events option is selected, the events can be generated by writing TMDAT and TMTAG core registers.

Each default trace scenario has a specific bandwidth. This indicates how many messages are routed in trace stream (on hardware), it depends on the trace scenario that you have selected to collect trace.

The following data can be viewed:

- Viewing Trace Data
- Viewing Timeline Data
- Viewing Critical Code Data
- Viewing Performance Data
- Viewing Counterpoints

### 3.5.2.1 Viewing Trace Data

Click the Trace link to view the trace data.

![Figure 3-47. Trace Viewer](image)

The below table explains the values of each counter for each trace scenario.

<table>
<thead>
<tr>
<th>Trace Scenario</th>
<th>Traid A</th>
<th>Traid B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Counter 0</td>
<td>Counter 1</td>
</tr>
<tr>
<td>Profiling - L2 cache events</td>
<td>Total L2 demand accesses</td>
<td>L2 program accesses</td>
</tr>
</tbody>
</table>

*Table continues on the next page...*
Table 3-15. Counter Values for each Trace Scenario (continued)

<table>
<thead>
<tr>
<th>Trace Scenario</th>
<th>Traid A</th>
<th>Traid B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Counter 0</td>
<td>Counter 1</td>
</tr>
<tr>
<td>Profiling - data</td>
<td>Data access L1 hits</td>
<td>Data access L1 pre-fetch hits</td>
</tr>
<tr>
<td>loads</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Profiling - clock</td>
<td>Application cycles</td>
<td>Inactive counter</td>
</tr>
<tr>
<td>cycles</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following table explains fields of trace data.

Table 3-16. Trace Data - Description of Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offset</td>
<td>First number represents trace message offset in raw trace file. Second number represents trace event index. One trace message can generate one or more trace events.</td>
</tr>
<tr>
<td>Event Source</td>
<td>Displays source of trace event</td>
</tr>
<tr>
<td>Description</td>
<td>Displays description of trace event</td>
</tr>
<tr>
<td>Call/Branch Source</td>
<td>Displays source function of trace event</td>
</tr>
<tr>
<td>Target</td>
<td>Displays target function of trace event</td>
</tr>
<tr>
<td>Type</td>
<td>Displays type of trace event</td>
</tr>
<tr>
<td>Timestamp (Cycles)</td>
<td>Displays the absolute clock cycles that the instruction takes to execute</td>
</tr>
<tr>
<td>Traid A</td>
<td>Displays number of events counted by counter 0 of triad A</td>
</tr>
<tr>
<td>Total L2 demand accesses</td>
<td>Displays number of events counted by counter 1 of triad A</td>
</tr>
<tr>
<td>L2 program accesses</td>
<td>Displays number of events counted by counter 2 of triad A</td>
</tr>
<tr>
<td>Triad B</td>
<td>Displays number of events counted by counter 0 of triad B</td>
</tr>
<tr>
<td>Total L2 hit</td>
<td>Displays number of events counted by counter 1 of triad B.</td>
</tr>
<tr>
<td>L2 instruction miss</td>
<td>Displays number of events counted by counter 0 of triad B.</td>
</tr>
<tr>
<td>L2 data miss</td>
<td>Displays number of events counted by counter 0 of triad B.</td>
</tr>
</tbody>
</table>

For more information on trace data, refer to the Viewing Trace Data topic.

3.5.2.2 Viewing Timeline Data

For information on Timeline data, refer to the Viewing Timeline Data topic.
3.5.2.3  Viewing Critical Code Data

For information on Critical Code data, refer to the Viewing Critical Code Data topic.

3.5.2.4  Viewing Performance Data

For information on Performance data, refer to the Viewing Performance Data topic.

3.5.2.5  Viewing Counterpoints

For information on counterpoints, refer to the Viewing Counterpoints in Analysispoints topic.

3.6  Exclude Symbols from Statistics

Exclude symbols feature allows you to configure the statistics before or after the data is processed.

The configure results dialog is available from:

- Software Analysis View
- Critical Code View
- Performance View
- Call Tree View

3.6.1  Software Analysis View

In Software Analysis view, the column Config Results provides a hyperlink to the result configuration view.

The configuration is saved in a .sym file inside the Analysis Data folder available at the project folder. Perform the following steps to configure the statistics before computing the results:
1. From the menu bar, select **Window > Show View > Other > Software Analysis > Software Analysis** to open the **Software Analysis** view.
2. After collecting the data, the data file will be listed in the **Software Analysis** view. The exclude symbols option will be available for each set of results.

![Software Analysis View](image)

**Figure 3-48. Software Analysis View**

3. Click the **Configure** hyperlink available under the **Config Results** column.

   The **Exclude symbols** dialog appears.
4. Select the libraries from the **Available items** list.
5. Click **Add**.

The selected libraries now appear under the **Selected items** list.
6. Click **OK**.

The configuration is saved with the list of all symbols and libraries that needs to be excluded from statistics.

**NOTE**

If the result configurator is opened for a new project, the latest saved configuration will be loaded into tables as a default configuration. If the configurator is not used, the results will be computed with all libraries and symbols.
You can also perform the following actions using **Exclude symbols** dialog:

### Table 3-17. Exclude Symbols Options

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter symbol</td>
<td>Acts as a real time filter that allows to search a symbol in a library from Available items list. The Available items list shows all the libraries that contain symbols with the same name available in the Filter symbol field.</td>
</tr>
<tr>
<td>Add custom item</td>
<td>Allows to add a customized symbol or an entire library under the Available items list.</td>
</tr>
<tr>
<td>Export item list to a file</td>
<td>Exports the configuration in a file. The file is saved in the Analysis Results folder of the current project location.</td>
</tr>
<tr>
<td>Import item list from file</td>
<td>Loads an existing configuration.</td>
</tr>
</tbody>
</table>

When scrolling through unprocessed data in the **Trace** viewer, the timestamp values and profiling counters values increases from the beginning of the trace, and at some point of time, the value gets reset. Also, the timestamp value does not increase beyond 16 million (approximately) and the profiling counters values beyond 256,000 (approximately). You can use the **Configure results** dialog to control the overflow of counters in the **Trace** viewer. Click the **Miscellaneous settings** tab, and select the **Enable accurate timestamp and profiling counters values for partially decoded trace** checkbox to enable the feature for handling timestamp overflow. When this checkbox is selected, the timestamps value count indefinitely, that is the values will not drop or reset while navigating through unprocessed data in the **Trace** viewer. If not selected, the behavior remains unchanged and the counters (timestamp and profiling) value increases till their maximum values and reset at some point.

Configure results has been enriched with the possibility to not store decoded trace while processing trace data for profiling viewers: Timeline, Critical Code, Performance, Call Tree and Counterpoints. By default processing trace data for these viewers is done with storing decoded trace. Processing trace data without storing decoded trace for profiling viewers is quicker than processing trace data with storing decoded trace, impact of storing or not decoded trace can be noticed for higher size of raw trace and executable. Select **Do not store decoded trace** option to disable storing of decoded trace.
3.6.2 Critical Code View

In Critical Code view, use the **Exclude symbols** icon to view the Exclude symbols dialog box.

Perform the following steps to view the configured statistics in Critical Code view:

1. Click the **Exclude symbols** icon available on the toolbar.
   
   The **Exclude symbols** dialog appears.

2. Select the existing configuration or modify the configuration by adding or deleting the libraries or symbols from the **Selected items** list.

3. Click **OK**.

   The configuration is applied on the Critical Code data. The old statistics will be deleted and the new statistics will be computed based on the configuration, and viewer will be repopulated with the new results.

Once the data is computed as per the statistics, the respective view will display the filtered data. The following figure shows an example that the func2 symbol is excluded from the computed data:
Exclude Symbols from Statistics

Figure 3-52. Critical Code View

3.6.3 Performance View
In Performance view, click the **Exclude symbols** icon available on the toolbar to filter out the symbols from the summary view table.

If the filtered symbol is called from another non filtered functions they will appear grayed in details table, in caller-callee pairs.

Figure 3-53. Performance View

### 3.6.4 Call Tree View

In the Call Tree view, click the **Exclude symbols** icon to filter out the symbols or libraries.

Filtered symbols would appear as grayed in the tree hierarchy.
3.7 Trace on Attach

Use trace on attach feature to help within a debug session of type Attach.

Following are the two main cases where this feature is useful:

- **On Application Crash Investigation**
- **On Changing Trace Configuration**

### 3.7.1 On Application Crash Investigation

Trace on attach launch configuration feature is a very handful way of investigating application crashes as it provides the retrieving of trace buffer that contains information on the context of the crash.

To perform this; enable the trace (program trace or profiling trace) and configure the trace buffer in overwrite mode in the moment of crash. For more information on configuring trace buffer, refer **Hardware Profiling**.

**NOTE**

Trace buffer can be configured in three different modes:
**Overwrite:** It provides trace just before the application crashes. It is considered as the most recommended mode of configuring trace buffer while investigating an application crash.

**One buffer:** It provides trace from the first time filled trace buffer and usually trace buffer gets filled before crash. It is recommended to use one buffer mode only if size of trace until the application reaches crash is small enough to fit in trace buffer or if trace buffer size is increased so that trace until application reaches crash fits into it.

**Continuous:** It is not recommended to configure trace buffer in continuous mode because in this mode every time trace buffer gets filled; core enters into a freeze state and waits for the trace buffer to be filled out. So if trace buffer is configured in continuous mode application will not reach crash section as core will be freeze waiting for trace buffer to be filled out.

Perform the following steps to investigate application crash using trace on attach feature:

- Create an Attach Launch Configuration
- Upload Trace
- Investigate Crash

### 3.7.1.1 Create an Attach Launch Configuration

Follow the steps given below to create an attach launch configuration:

1. In the CodeWarrior Projects view, right-click on the project that was crashed and select **Debug as > Debug Configurations** option from the context menu.
2. In the Debug window, select **CodeWarrior** configuration and select **New** from the context menu.
3. On the **Main** page, select **Attach** as the **Debug session type**.
4. Specify the project name that crashed in the **Project** field. If the application is not displayed in the **C/C++ Application** field, click **Search Project** to select the application image.
5. Click **Debug**.

**NOTE**

By default, the **Trace control settings** will be set on **Manually (Using trace buttons from Trace Commander)** option while using the **Attach** mode.
configuration. Hence, the trace collection will not start automatically. To enable the trace collection, use the trace control button in the **Trace Commander** view.

### 3.7.1.2 Upload Trace

Click on the **Upload Trace** button available in the Trace Commander view to retrieve the trace buffer from target hardware to host machine.

The **Upload Trace** button gets enabled every time whenever the **Trace collection is ON**, therefore the trace can be uploaded even when trace has already been collected at a suspend state (when debug session is not in the attach mode). Following are the effects while uploading trace data after trace has already been collected:

1. If the trace buffer is configured in **One buffer** mode or **Overwrite** mode and the upload trace button is hit to upload the trace data then the already collected trace data will be removed. The trace buffer will be read once again and trace data should look same as before hitting the upload trace button.
2. If trace buffer is configured in **Continuous** mode and upload trace button is hit then the trace data already collected is removed, the trace buffer will be read once again and the trace will become empty (reset the trace buffer in the continuous mode to unfreeze the core).

### 3.7.1.3 Investigate Crash

Inspect trace in Trace viewer towards its end.

The following figure displays the trace data in Trace viewer when an application crashes and when trace mode has been set to Program trace. For more information on how trace mode can be set to Program trace or Profiling trace, refer **Hardware Profiling**.
### Figure 3-55. Cash Investigation in Trace Viewer using Program Trace

The following figure shows the trace data in Trace viewer when an application crashes and when trace mode has been set to profiling trace.
Tip

Exceptions can be easily recognized in Trace viewer by their orange color.

3.7.2 On Changing Trace Configuration

Trace on attach feature can also be used to change trace configuration on board when a process is already running on it.
For instance you can attach to the running process and enable trace if it was not enabled before or you can change from program trace to profiling trace. To change the trace configuration using trace on attach feature, perform the following steps:

- Create an Attach Launch Configuration
- Change Trace Configuration

**3.7.2.1 Create an Attach Launch Configuration**

To create an Attach launch configuration, refer Create an Attach Launch Configuration.

**3.7.2.2 Change Trace Configuration**

Perform the following steps after creating the attach launch configuration:

1. Suspend the debug session to configure trace.
2. Click **Configure Trace** icon available under the **Trace configure** column in the Trace Commander view.
   
   The **Configure Trace** dialog box appears.
3. Select **Program trace** option from the **Trace scenarios** list.
4. Click **OK**.

The new configuration gets applied on the target when debug session is resumed.

**3.8 Viewing Interrupts**

The interrupts in an application flow can be easily recognized in Trace viewer with their 'Orange' color and their **Type** flag.

The following figure shows an example on how interrupts are displayed for Program trace in Trace viewer:
The below figure shows an example on how interrupts are displayed for Profiling trace in Trace viewer:

Figure 3-58. Profiling Trace Interrupts Displayed in Trace Viewer
3.9 Preparing Aurora Interface for Collecting Aurora Nexus Trace

This topic describes how to prepare Aurora interface to extract Nexus trace from the Aurora port (for processors that support Aurora) using a Gigabit tap with Aurora daughter card.

If you want to collect Nexus trace from the target using Aurora, you need to first prepare the Aurora interface to be used by the trace probe. A TCL script or the Aurora training script is used to train Aurora on the target. This topic contains the following sub-topics:

- Enabling Aurora
- Executing Aurora Training Script

3.9.1 Enabling Aurora

You can enable Aurora on B4860QDS board using Reset Configuration Word (RCW). For B4860QDS boards, Aurora lanes need to be properly routed to Aurora connector on the board. This is done automatically from u-boot, when a RCW with Aurora enabled is detected.

When Aurora is enabled, the Aurora interface needs to be trained so that it can run properly.

3.9.2 Executing Aurora Training Script

The Aurora training script should be executed before configuring or extracting trace through the Aurora port. You can execute the training script to train Aurora on B4860 targets as follows.

For B4860QDS target, the TCL script that is used to train Aurora on the target is available in the CodeWarrior installation folder at:

`SC/StarCore_Support/Initialization_Files/Aurora/train_aurora_b4.tcl`

The routine inside this file which executes the required training is called `train_aurora`. 
To train Aurora on the B4860QDS target:

1. Ensure that the RCW on board is having Aurora enabled on SERDES#1 and the speed is set to 2.5 Gbps. You can modify RCW fields for different SERDES protocol values to have Aurora enabled, for example:
   - For bits 128-134 in RCW, configure Serdes#1 protocol. Make sure that protocol selected is 0x30. It will enable 2 Aurora lanes.
   - For bit 176 in RCW, configure Aurora divisor. Make sure this bit has value 1, so that Aurora operates at 2.5Gbps.

You can write a new RCW to target using the CodeWarrior Flash Programmer utility.

2. Execute u-boot. A proper u-boot detects that Aurora is enabled in RCW and initializes cross-points for routing Aurora signals to Aurora connector. If you do not have an u-boot or you have an u-boot without Aurora routing support, you can write it using Flash programmer from CodeWarrior.

   **NOTE**
   For more information on writing u-boot using Flash programmer, refer to *CodeWarrior Common Features Guide.pdf* in the folder: `<CWInstallDir\SC\Help\PDF`, where `<CWInstallDir>` is the directory where CodeWarrior for StarCore V10.x is installed.

3. Run the TCL script from the CCS console window or the **Debugger Shell** window.
   - From the CCS console window:
     1. Make sure you have a proper Command Converter started and configured.
     2. In console, type: `source {$path}/train_aurora_b4.tcl`, where `$path` is a valid system path to the TCL script.
     3. In console, type: `train_aurora` to call the training procedure.
     4. Look for a confirmation message that Aurora interface was successfully trained. If it was not successfully trained, repeat step c.
   - From the **Debugger Shell** window:
     1. In shell, type: `source {$path}/train_aurora_b4.tcl`, where `$path` is a valid system path to the TCL script.
     2. In shell: `train_aurora_dbg` to call the training procedure.
     3. Look for a confirmation message that Aurora interface was successfully trained. If it was not successfully trained, repeat step b.

   **NOTE**
   A target reset after u-boot initialization for Aurora will result into a link down. This means that Aurora interface cannot be used anymore. If you...
use CodeWarrior for downloading the projects on the target, make sure you have unchecked the option to reset the target. If you need to reset the target, ensure after reset that u-boot has a chance to run and initialize routing signals for using Aurora (estimated time is 10 seconds). You can use an initialization script that can handle a target already initialized by u-boot.

**NOTE**

If you see HBDP errors in the collected trace, it is useful to run train_aurora_dbg in the Debugger Shell window. This helps in eliminating most of the HBDP errors from trace.

### 3.10 Inline Functions

Inline functions are used to optimize the source code. It is typically used for functions that execute frequently.

Perform the following steps to add inline function:

1. Create a Stationary project for SC3900 PACC.
2. Add inline function in source file, sc3900_main.c. The following example is taken:
   
   ```c
   int inline_func(int i){#pragma inline;i++;i+=200;return i;}
   ```

3. Select the project and right-click on it. From the context menu, select **Properties**. The **Properties** dialog box appears. Select **C/C++ Build -> Settings -> Starcore 3900 C/C++ Linker Application -> Linker Settings**. Uncheck **Strip Dead Code** checkbox.
4. Build the project.
5. Open Debug Configurations dialog box and check **Enable Trace and Profile** checkbox in the **Trace and Profile** tab.
6. Click **Apply** and debug the application.
7. Resume debug session and then suspend the debug session after sometime.
8. To open Software Analysis view, click **Windows -> Show View -> Software Analysis**.
9. Check for inline function in the Trace, Timeline, Critical Code, Performance, Call Tree viewers.
NOTE

On B4860 SC3900 targets, the only profiling feature providing support for inline functions is Critical Code. Timeline, Performance and Call Tree features do not provide support for inline functions because of hardware trace limitation. Profiling trace messages used by these features are being generated only on call and return instructions. Since body of inline functions is embedded into parent function that calls inline function, trace profiler features based on functions calls are not aware of inline functions.

In Trace data viewer, inlined functions are displayed as <function_name_address>, for example, inline_func_0x40002bd2.

Figure 3-59. Trace Data with Inline Function

In Timeline data viewer, inlined functions are represented in a different color than default one. For example, inlined function details are shown in blue.
In Critical Code data viewer, inlined functions are suffixed with `inline`.
In Performance data viewer, inlined functions are suffixed with `inline`.

In Call Tree data viewer, inlined functions are suffixed with `inline`.
3.11 Importing Offline SC3900 Trace with Hardware Trace Configuration

This feature allows you to import offline SC3900 trace that is configured with hardware profiling.

Before importing SC3900 trace, it is important to know that profiling triads were active on board or not when trace was collected.

There are 2 triads (each triad is a group of 3 profiling counters) that are used when trace is imported, triad A and triad B.

The profiling triads should be selected while importing, only if trace contains values for selected counters. If trace does not contain values for selected counters, there is no need to select profiling triads while importing the trace.

To import offline SC3900 trace data into your project:

1. Select **File > Import** to open the Import wizard.
2. Expand the **Software Analysis** node and select **Trace**.

![Figure 3-63. Call Tree Data with Inline Function](image)
3. Click **Next** to display the **Import Trace** page of the Import wizard.
4. Click **Browse** and locate the trace data file of the project that you want to import into your project. The trace data file is located in the `.Analysis Data` folder of the project in the workspace.

5. If you select the **View the trace data on finish** checkbox, the imported trace data will automatically open in the **Trace Data** viewer of your project. If you clear this option, the imported data will be visible from the **Software Analysis** view. The imported trace data is also saved in the `.Analysis Data` folder of your project.

6. Click **Next** to display the **Import Trace to Existing Project** page.
Figure 3-66. Import Wizard - Import Trace to Existing Project Page

7. Type or select the project in which you want to import the trace data.
8. Type a new file name for the trace data file in the New trace file name will be text box. This field is optional.
9. Click Next to display the Import Trace Configuration page. This page allows you to specify the trace configuration details of the project to display imported trace data based on the executable file or launch configuration used during trace collection.
10. Select the target of your project from the **System** drop-down box.
11. Select the desired core from the **System** list.
12. Left click in the corresponding **Trace Settings** group, and click ![image]
   The **Trace Settings** dialog appears.
13. Select any of the following options from the **Trace Settings** group:

- **None** if you do not want to correlate trace data with the source of your project, that is when the trace being imported is not program trace. The trace data will appear without displaying the source lines corresponding to the trace events.
- **Application** if you want to correlate trace data with the source. Click **Browse** to locate the executable file (.abs/.elf/.afx) of the application used during trace collection.
- **Launch Configuration** to select the launch configuration of the application used during collection to identify the executable file or processor/system of the application.

14. Check **Profiling trace** option available under the **Trace type** list.

![Figure 3-69. Trace Settings dialog - Trace type](image)

15. Select the **Profiling Triads** if the trace that you are importing contains the counter values. The counters of both triads have default name. To change the name of the counters of a triad, select the triad checkbox and the **Edit** button is enabled. Click **Edit** to rename the counters. Legal characters for counter name are A-Z, a-z, ., _, - and whitespace. First legal character is A-Z or a-z. Also, names of counters should be distinct.

16. Click **OK**.

17. Click **Finish**.
If trace data file contains values for Triad A, only Triad A was configured during trace collection and **Profiling - clock cycles** scenario was selected. You should select same counters while importing trace data. The trace viewer displays the 3 counter values of triad A.

![Figure 3-70. Trace Data with Triad A Counter Values](image)

The trace data file contains profiling messages for 6 counters (values of triad A and B) if both triads were configured during trace collection and trace scenario **Profiling - L2 cache events** was selected. While importing, if only triad A (3 counters) is selected, trace viewer will not display any counter value (as shown below). Number of triad counters within profiling messages from trace data file must match with number of triad counters selected at the time of importing for trace viewer to display counter columns.
NOTE
Trace viewer displays trace containing no counter values if the trace was collected using **Program Trace** configuration.

![Figure 3-71. Trace Data with No Counters](image)

You should select both triad A and B while importing, if both triads were configured during trace collection when trace scenario **Profiling -L2 cache events** was selected.
Chapter 3 Collecting Data

3.12 Multicore Tracing

This section explains the various actions performed on the multicore trace data.

Before debugging a multicore project and collecting trace data on the B4 target hardware, create and build a B4 project with Trace and Profile enabled.

NOTE
To successfully apply multicore trace configuration, specify or set the platform configuration (default one, a new one or one already created) for each of the launched configuration using Debug Configurations window. If launch configuration is not edited, then the platform configuration is not set. Hence, no trace data is collected for the core associated with that launch configuration.
This section includes the following:

- Debugging Multicore Project
- Collecting Multicore Trace Data
- Viewing multicore trace data
- Importing multicore trace data
- Trace commander view
- Multicore continuous trace

### 3.12.1 Debugging Multicore Project

This section explains how to debug a multicore trace project.

To debug a multicore project:

1. Right-click the B4860 project in the CodeWarrior Projects view and select **Debug As > Debug Configurations**.

   The **Debug Configurations** dialog box appears.

2. Double-click the Launch Group node in the tree structure on the left.

   The **New_configuration** node is added.
3. Type a name for the new launch group in the **Name** text box, for example, **Test_Multicore_TraceData_cores**.
4. Click **Add**.

   The **Add Launch Configuration** dialog box appears.

   Expand the **CodeWarrior Download** node, and select the first two cores with **Shift** key pressed.
5. Click **OK**.

The cores get added to the new launch group.
NOTE
The cores selected to be part of the launch group should be configured before debugging the launch group.

6. Click **Apply** to save the settings.
   
   Click **Debug** to start the debug session. The Debug perspective appears and the execution halts at the first statement of `main()`.
3.12.2 Collecting Multicore Trace Data

This section explains the ways to collect multicore trace data.

Follow the steps to collect multicore trace data:

1. In the Debug view, click Multicore Resume. The execution begins and data measurement starts. Wait for some time.

2. When output starts displaying in the Console view for all the two cores, click Multicore Terminate.

3.12.3 Viewing multicore trace data
The trace data is collected in the Software Analysis view. Click on the respective hyperlinks to view the collected data.

**NOTE**

CriticalCode link is available only when all cores that are active in a debug session are configured with Program Trace scenario. Whereas, Timeline, Performance, and Call Tree links are available only when cores are configured with Profiling scenario.

**NOTE**

If the launch group contains cores configured with both Profiling and Program Trace, then only the Trace link will be available in Software Analysis view.

The following figure shows an example of Software Analysis view:

![Figure 3-77. Software Analysis View](image)

Click on the Timeline link to view the timeline data.
Click on the Critical Code link to view the critical code data.
Figure 3-79. Critical Code View

Click on the Performance link to view the Performance data.
Figure 3-80. Performance View

**NOTE**

The caller-callee pie chart will move according to the function pressed in details table and exclude symbols will exclude the selected symbols for all cores.

Click the Call Tree link to view the call tree data.

Figure 3-81. Call Tree View
3.12.4 Importing multicore trace data

This feature allows you to import the trace data collected for multicore B4 hardware project.

To import the multicore trace data, perform the following steps:

1. Collect the multicore trace data.
2. Switch to a new workspace.
3. Select **File > Import** to open the Import wizard.
4. Expand the **Software Analysis** node and select **Trace** option.
5. Click **Next** to display the **Import Trace** page of the **Import** wizard.
6. Click **Browse** and locate the trace data file of the project that you want to import into your project. The trace data file is located in the `.Analysis Data` folder of the project in the workspace.
7. Select either **Existing project** or **New project** from the **Import To** list, depending upon the requirements of your project.
8. Click **Next** to display the **Import Trace to Existing Project** page.
9. Type or select the project in which you want to import the trace data. Type a new file name for the trace data file in the **New trace file name** will be text box. This field is optional.
10. Click **Next** to display the **Import Trace Configuration** page.
11. Select the **System** corresponding to the trace that will be imported.
12. Specify the **Source Code Correlation** details for the cores on which trace was generated, for example SC3900-0 and SC3900-1.
13. Specify the **Application** or launch configuration and the respective **Trace type** for each core in the **Source Code Correlation** dialog box.
14. Click **Finish**.
15. The trace viewer will display the imported multicore data.
The following image shows the imported multicore trace data with profiling trails.

![Image of imported multicore trace data](image-url)

**Figure 3-82.** Imported multicore trace data

![Image of imported multicore trace data - Traid counter values](image-url)

**Figure 3-83.** Imported multicore trace data - Traid counter values

### 3.12.5 Trace commander view

[CodeWarrior Development Studio for StarCore SC3900FP DSP Architectures Tracing and Analysis Tools User Guide, Rev. 10.9.0, 11/2015]

120 Freescale Semiconductor, Inc.
Trace commander view is used to manage the multi-core trace collection data. The view is used to perform actions such as, starting or stopping the tracing on different cores, manual upload, trace configure or changing the master core of the collected trace data.

Trace commander displays current trace state for each core.

Perform the following steps to manage the multi-core trace data using a trace commander:

1. Create a multi-core project.
2. Right-click the project and select **Debug As > Debug Configurations**.
   
   The **Debug Configurations** dialog box appears.
3. Create a **New_Launch** group by adding the cores on which trace needs to be collected, for example core 00 and core 01.
4. Click **OK**.
   
   The cores get added to the new launch group.
5. Set the **Trace control** settings for the selected cores on which multi-core debugging needs to be performed. The trace control settings are available on the **Intermediate** page of **Trace and Profile** tab.
6. Click **Apply**.
7. Debug the **New_configuration** group to collect the trace data.
8. Select **Window > Show View > Other > Software Analysis > Trace Commander**.
   
   The **Trace Commander view** is displayed.

![Trace Commander View](image_url)

**Figure 3-84. Trace Commander View**

The following table lists the description of each column available in the **Trace commander** view:
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform Configuration</td>
<td>Shows the platform configuration file used by projects.</td>
</tr>
<tr>
<td>Data stream</td>
<td>Shows the current data stream selected in configurator.</td>
</tr>
<tr>
<td>Trace generator</td>
<td>Displays the current cores that are available for collecting trace.</td>
</tr>
<tr>
<td>Master trace generator</td>
<td>Displays the current core selected as the master core and allows changing the master core.</td>
</tr>
<tr>
<td>Trace control</td>
<td>Allows you to start/stop and upload the trace data. The <strong>Upload</strong> button is enabled only when trace is available for collection.</td>
</tr>
<tr>
<td>Trace status</td>
<td>Displays the current trace status.  <code>N/A</code> is applicable when cores are not in the debug state.  <code>Trace stop</code> or <code>Trace start</code> appears when cores are in the debug state, depending on the trace state for that core.</td>
</tr>
<tr>
<td>Trace configure</td>
<td>Allows you to configure the trace settings.  <strong>NOTE:</strong> If trace configuration settings are applied manually, then after reconfiguring the trace using <strong>Trace configure</strong> button, Start/Stop trace button corresponding to master core will be reset to the <strong>Start – Trace is OFF</strong> state and the Upload button will be disabled if it was previously enabled.</td>
</tr>
</tbody>
</table>

9. Click the **Trace** link available in the **Software Analysis** view to view the trace collected on the respective cores.
The trace viewer displays the trace collected for core00 and core01.

### 3.12.6 Multicore continuous trace

Multicore continuous trace stands for trace collected from multiple cores without trace being lost. For the B4860 targets, trace might be lost for two main reasons:

- Trace buffer size is not enough to accommodate all trace generated by the multicore trace session.
- Trace messages frequency/rate, even if trace buffer size is enough to accommodate all trace generated by the multicore trace session.

In order to avoid trace messages lost in any of the above cases trace collection mode should be set to continuous.

To collect the profiling trace in the multicore continuous trace mode, follow the steps listed below:

1. Create a project using b4860 hardware target.
2. Use source `multicoreContinousTrace.c` with:
   - `#define FULL_PROGRAM_TRACE`
   - `#define TRACE_MES_FREQ_TRACE_MES_FREQ_HIGH within #ifdef PROFILING_TRACE block`. 

---

**Figure 3-85. Trace Data Collected**

The trace viewer displays the trace collected for core00 and core01.
3. Select Profiling – L2 cache events trace scenario for all cores.
4. Select DDR trace buffer size to 0x60000 and one buffer as trace collection mode for master core.
5. Start multicore debug and wait until application reaches its end. Multicore terminate.
6. Open Performance to fully decode trace and store decoded trace.
7. Open **Trace viewer** and press **Ctrl+F** to find *error*.
8. Inspect Performance viewer.
9. Select DDR trace buffer size to 0x60000 and continuous as trace collection mode for master core.
10. Start multicore debug and wait until application reaches its end. Multicore terminate.
11. Open Performance to fully decode trace and store decoded trace.
12. Open **Trace viewer** and press **Ctrl+F** to find *error*.
13. Inspect Performance viewer.
14. Redo steps 10 to 13 with:
   - DDR trace buffer size set to 0x10000 and continuous as trace collection mode for master core.
15. Redo steps 10 to 13 with:
   - DDR trace buffer size set to 0x60000 and one buffer as trace collection mode for master core.
   - TRACE_MES_FREQ set to TRACE_MES_FREQ_MEDIUM within #ifdef PROFILING_TRACE block. Rebuild the project.
16. Redo steps 10 to 13 with:
   - DDR trace buffer size set to 0x10000 and continuous as trace collection mode for master core.
   - TRACE_MES_FREQ set to TRACE_MES_FREQ_MEDIUM within #ifdef PROFILING_TRACE block. Rebuild the project.
Chapter 4  
Setting Tracepoints

Tracepoint is a point in the target program where start or stop tracepoints are set at a line of the source code or the assembly code. The start and stop tracepoints are triggers for enabling and disabling the trace output. The advantage of setting start and stop tracepoints is to capture the trace data from the specific part of the program. This solves the problem of tracing a large application. A full trace would be extremely difficult to follow and would also require a large amount of target memory to store it. The trace data displays the trace result based on the tracepoints. If there is a stop tracepoint and no start tracepoint, the trace automatically starts at the beginning of the application. If there is at least one start tracepoint, the trace is automatically disabled at the beginning of the application. This chapter explains how to set start and stop tracepoints and also how to enable and disable a tracepoint.

This chapter explains:

- Setting Start and Stop Tracepoints
- Enabling and Disabling the Tracepoints
- Limitation of hardware tracepoints
- Multicore hardware tracepoints

4.1 Setting Start and Stop Tracepoints

This section explains how to start and stop tracepoints.

The tracepoint can be set in the Editor area and the Disassembly view. The source line tracepoint is set in the Editor area and the address tracepoint is set in the Disassembly view. You can see the tracepoints in the Editor area as well as in the Disassembly view.
NOTE

It is recommended that start and stop tracepoints should be set in the same function so that the trace that is collected is meaningful.

This topic explains how to start and stop tracepoints using the following:

- Simulator
- Target Hardware
- Viewing Tracepoints in Analysispoints

4.1.1 Simulator

This section explains how to set tracepoints in source code of a simulator project.

Before setting the tracepoints, refer to the topic Collecting Data for the procedure on how to collect the trace and profiling data using SC3900 simulator.

Perform the following steps to set the tracepoints in source code:

1. In the CodeWarrior Projects view, select the Source folder under the project.
2. Double-click on the source file, for example, sc3900_main.c. The contents of the file can be viewed on the right-side of the pane.
3. Select the line on which you want to set the start tracepoint.
4. Right-click on the marker bar, and select Toggle Trace Start Point> Hardware Trace Point option from the context menu as shown below. The same option is also used to remove the start tracepoint from the marker bar.
5. Select the line on which you want to set the stop tracepoint, right-click on the marker bar, and select **Toggle Trace Stop Point** > **Hardware Trace Point** option from the context menu. The same option is also used to remove the stop tracepoint from the marker bar.

**NOTE**

The mouse pointer over a tracepoint icon displays the attributes of tracepoints on that line.
Figure 4-2. Setting Stop Tracepoint

NOTE

The start tracepoint is in green color and stop tracepoint is in red color. Do not set tracepoints on the lines containing only comments, brackets and variable declaration with no value.

6. After setting the tracepoints, debug the application and collect the data in the datafiles. The datafiles are generated by the simulator in which the data has been collected before setting the tracepoints in the source code. In this datafile, the Description field shows that the main function is called first and then it calls the func2 function.
Figure 4-3. Trace Data Page Before Setting Tracepoints

In this datafile, after setting the start and stop tracepoints as displayed in the above figures, the **Description** field shows that the main function calls the func2 and the tracing starts from func2.

---

Figure 4-4. Trace Data Page After Setting Tracepoints

In the critical code data, you can see that tracing has started from func2.
Similarly, you can view the other datafiles. For details, refer to the Viewing Data topic.

You can also set the tracepoints in the Disassembly view when the debug session is active. To get the Disassembly view, select Windows > Show View > Disassembly. (Debug perspective) The Disassembly view displays the C/C++ program in assembly language. When you set the tracepoints in Disassembly view, the tracepoints are automatically displayed in the Editor area.

To set the tracepoints in Disassembly view, follow the same steps as you followed for Editor area.

The tracepoint attributes of Source Code and Disassembly view are displayed in the Analysispoints view. To get the Analysispoints view, select Windows > Show View > Other > Software Analysis > Analysispoints.

### 4.1.2 Target Hardware

This section explains how to start and stop hardware tracepoints.

For B4 boards, there are 3 start hardware tracepoints and 3 stop hardware tracepoints. If hardware tracepoints exceed this limit, only 3 start tracepoints and 3 stop tracepoints will control the trace, the other tracepoints will be ignored.
If only one stop hardware tracepoint is used then application will be traced from its beginning until tracepoint. Also, if only one start hardware tracepoint is used then application will be traced from tracepoint until the end of the application. When more than one hardware tracepoints are used then they should be used in pair, for each start hardware tracepoint set, a stop start hardware tracepoint should be set as well. A start hardware tracepoint marks the start of traced section while a stop hardware tracepoint marks the end of traced section.

Hardware tracepoints control only program trace and profiling trace, they have no effect on any other kind of trace, for example, debug messages, or user defined events.

**NOTE**

Start and stop tracepoints should be set on C/C++ or assembly linear instructions to ensure all change of flow instructions between start and stop are traced. If start and stop tracepoints are set on C/C++ or assembly change of flow instructions then these instructions are not traced.

**NOTE**

Hardware or software breakpoints and stepping should not be used inside the traced section in order to obtain accurate trace and profile results. The reason of this behavior is that on single stepped VLES, the events generated in parallel like trace generation, address detection, and event counting are ignored. For experts only, trace and profile results can be accurate even though hardware or software breakpoints or stepping are used inside the traced section only if they are set such that none of single stepped VLES-es to interfere with the already mentioned events.

To set the hardware tracepoints in source code:

1. In the **CodeWarrior Projects** view, select the **Source** folder of b4860 under the project.
2. Double-click on the source file, for example, b4860_main.c. The content of the file can be viewed on the right-side of the pane.
3. Select the line on which you want to set the start hardware tracepoint. For example, set start tracepoint on C linear instruction before `func2` call as shown in figure below.
4. Right-click on the marker bar, and select the **Toggle Trace Start Point> Hardware Trace Point** option from the context menu.
5. Select the line on which you want to set the stop hardware tracepoint. For example, set stop tracepoint on C linear instruction after `func2` call as shown in figure below.
6. Select the line on which you want to set the stop hardware tracepoint, right-click on the marker bar, and select **Toggle Trace Stop Point > Hardware Trace Point** option from the context menu.

![Image of C code](image)

**Figure 4-6. Setting Start and Stop Tracepoints**

7. In the **Trace and Profile** tab of **Debug** window, select **Program trace** scenario.
8. Click **Debug** and wait until application reaches its end.
9. Terminate the application.
10. Inspect trace viewer and available profiling viewers.
The above figure shows the datafiles generated by the target hardware in which the data has been collected after setting the tracepoints in the source code. Tracing begins with a call from `main` to `func2` and ends with return `func2` to `main`.

If a B4 StarCore project contains a launch configuration with SC3900 simulator set as target, the hardware tracepoints will have no effect on trace while running the project on a B4 board. Perform the first two steps if tracepoints are already added on a source file from a project, otherwise skip to point 3 onwards if tracepoints have never been added on a source file:

1. Select `Window > Show View > Other > Software Analysis > Analysispoints`, the Analysispoints view is displayed. In the Analysispoints view, remove all the analysispoints.
2. Remove the `SaAnalysispointsManager.apconfig` file from the project folder for which the trace data has been collected.

   **NOTE**
   
   Project files are saved in `<workspace location >/< project >` folder.

3. Rename the extension of any `* .launch` file from the project location for which the target has been selected as SC3900 simulator.

   **NOTE**
   
   The launch files are saved in `<workspace location >/< project >/< Debug_Settings >` folder.
4. Select the existing project from **CodeWarrior Projects** view.
5. Right-click the project and select **Refresh** menu.
6. Add tracepoints on the source file of the selected project.

### 4.1.2.1 Modifying Hardware Tracepoint Configuration

This section explains how to modify the configuration of hardware tracepoints.

You can change the hardware tracepoints configuration during the debug session, such as adding, removing, enabling, disabling or ignoring a tracepoint. You will be notified when changes are made to hardware tracepoints configuration, but they are not applied to the current debug session.

![Figure 4-8. Tracepoints Configuration Changed](image)

This message box appears at every resume if tracepoints configuration change is made and can not be applied to current debug session. If you have selected **Don't prompt again** checkbox then you will not be notified.

You can activate tracepoints configuration change notification by not selecting **Don't prompt to notify tracepoints configuration changed during debug session** in the Preferences -> Software Analysis window.
4.1.3 Viewing Tracepoints in Analysispoints

Use the Analysispoints view to view the attributes of tracepoints.

This view display the attributes of tracepoints set in source code or/and assembly code. To view the attributes of tracepoints, (Debug perspective) select Window > Show View > Other > Software Analysis > Analysispoints, the Analysispoints view is displayed. The source line tracepoint is set in the Editor area and the address tracepoint is set from the Disassembly view.

4.1.3.1 Disassembly View

This section lists the address tracepoint attributes set in Disassembly.

Following are the address tracepoint attributes set:

- Type of tracepoint (Software or Hardware)
- File name
Setting Start and Stop Tracepoints

- Address of the instruction of assembly code
- Action (Start or Stop)

4.1.3.2 Editor Area

Lists the source code tracepoint attributes set in the Editor area.

Following are the source code tracepoint attributes set in the Editor area:

- Type of tracepoint (Software or Hardware)
- File name
- Line number
- Action (Start or Stop)

You can also enable and disable tracepoints from this view. To enable or disable a particular tracepoint, right-click on the attribute and select **Enable Tracepoint** or **Disable Tracepoint** option from the context menu.

![Analysispoints View](image)

**Figure 4-10. Analysispoints View**

The **Analysispoints** view shows you where start and stop tracepoints are set in the Editor area or the **Disassembly** view. It helps you to navigate to the source code or assembly code where the tracepoint is set. If you want to go to the specific line of a source file where the tracepoint is set, click the **Go To File for Analysispoints** icon. This option will help you to navigate to that line.

If you want to ignore all the tracepoints at the time of execution, click the **Ignore All Analysispoints** icon. To remove a tracepoint or remove all the tracepoints, click **Remove Selected Analysispoint** icon or **Remove All Analysispoints** icon.

To view the complete path of the files of the attribute, click the **View Menu** of **Analysispoints** view and select **Show Full Paths** option.
You can also group the attributes by files or projects. To group the attributes by files, click **ViewMenu > Group By > Files** option. The attributes are grouped by files as shown below.

To group the attributes by projects, click **View Menu > Group By > Projects** option. The attributes are grouped by projects as shown below.

To ungroup the attributes, click **View Menu > Group By > Analysispoints** option as shown below.
4.2 Enabling and Disabling the Tracepoints

You can also enable and disable tracepoints using this Analysispoints view.

If you want to enable the Start or Stop tracepoint, right-click on the marker bar where Start or Stop tracepoints are already set and in disabled state, select the **Enable Tracepoint** option from the context menu.

If you want to disable the Start or Stop tracepoint, right-click on the marker bar where Start or Stop tracepoint is already set and enabled, select the **Disable Tracepoint** option from the context menu. A disabled tracepoint will have no effect during the collection of trace data. You can also disable the tracepoint from Analysispoints view. Right-click on the selected attribute and select **Disable** option. The unchecked attribute indicates the disabled tracepoint.

4.3 Limitation of hardware tracepoints

This section explains the various limitations of hardware tracepoints.

B4 SC3900 core has limited number of hardware resources allocated for hardware tracepoints. You can define upto 3 start hardware tracepoints and 3 stop tracepoints, but if the number of hardware tracepoints defined by user exceeds the number of defined tracepoints, then a notification is raised. The notification message specifies:

- the core id on which installation has failed
- the hardware tracepoints that are not successfully installed on target are identified through line, source file pair or address
the reason for installation failure
the total number of start/stop hardware tracepoints tried to be installed and number of them for which hardware resources were found free, only if install has failed for start/stop hardware tracepoints category

However, install is performed only on tracepoints that are enabled. The hardware tracepoints that are reported with install failure does not have any effect on trace in the ongoing debug session.

It is also possible that some of the hardware resources allocated for hardware tracepoints are already reserved or in use by some other module, for example, debugger is another module that can use these resources for hardware breakpoints.

Let us consider an example of hardware tracepoints notification failure, where start and stop hardware line tracepoints exceeds the maximum number of hardware resources available:

1. Create a b4860 hardware project. For details, refer to Hardware Profiling
2. Open the source file, b4860_main.c.
3. Set four start and stop tracepoints at the beginning and end of the functions in b4860_main.c file.
4. Build the project.
5. Open Debug Configurations dialog and select Program trace option available under the Trace scenarios list on the Basic tab page of Trace and Profile tab.
6. Click Apply and debug the application.
7. Resume the debug session and wait until application reaches its end.

The Trace configuration failure dialog gets prompted, while the debug application keeps running till its end.
8. Open Software Analysis view and inspect the trace viewer.

The trace viewer shows the trace collected according to the successful installation of hardware tracepoints. If there is at least one start tracepoint successfully installed then trace is considered to be stopped, and will be started when first start tracepoint is hit.

### 4.4 Multicore hardware tracepoints

This section explains the support of hardware tracepoints for B4 SC 3900 on multicore.

For each core:

- There are up to 3 start hardware tracepoints and 3 stop hardware tracepoints available
- Hardware tracepoints control only program and profiling trace modes and have no effect on other trace modes
- When number of hardware tracepoints set exceeds above limitation user notification is raised

When a tracepoint is added, it is by default set (enabled) on all the available cores of the platform. For example, if the target is B4860 then the tracepoint is set on all the six cores of SC3900.

#### 4.4.1 Set cores
The Set cores feature is used to handle hardware tracepoints on multicore.

It is available for each hardware tracepoints.

Follow the steps given below to set hardware tracepoints on multicore:

1. Select **Window > Show View > Other > Software Analysis > Analysispoints**, the Analysispoints view is displayed.
2. Right-click on the tracepoints listed in the Analysispoints view and select **Set cores** option.

   The **Select cores** dialog box appears.

   ![Select cores dialog](image)

   **Figure 4-16. Select cores dialog**

3. Select the cores on which hardware tracepoint needs to be enabled or disabled.
4. Click **OK**.

   The list of cores on which tracepoint is enabled gets added in Analysispoints view along with type and action performed.

   ![Selected cores displayed in Analysispoints view](image)

   **Figure 4-17. Selected cores displayed in Analysispoints view**
Also when start and stop hardware line tracepoints exceeds the maximum number of hardware resources available, the hardware tracepoints failure notification message will be prompted for each individual core.
Chapter 5
Setting Counterpoints

Counterpoints allow you to count events between two points of the executed code. The results are computed based on the collected trace data; the counterpoints are searched in trace by their addresses. When counterpoints are found in the executed code, the delta statistics is computed between two counterpoints. Counterpoints are only supported in Profiling trace.

This chapter explains:

- Setting Counterpoints
- Viewing Counterpoints in Analysispoints
- Enabling and Disabling the Counterpoints

5.1 Setting Counterpoints

This section explains setting the source line counterpoint in the Editor area and the address counterpoint in the Disassembly view.

The counterpoints are set similarly as tracepoints.

Both types of counterpoints have associated address:

- For Disassembly counterpoints, it is the address of the assembly code where the counterpoint is placed in the Disassembly view.
- For source line counterpoints, it is the address of the source line where the counterpoint is placed. The address is determined at runtime.

You can see the attributes of counterpoints from the Analysispoints view.
The counterpoints results are computed based on the traced data; you must place the counterpoints on the code that generates the trace. If the counterpoints are set on the lines containing only comments, brackets and variable declaration with no value, a **Found invalid counterpoints** dialog box gets prompted as shown below.

![Found invalid counterpoints dialog box](image)

You can either disable the counterpoint or move the counterpoint to location suggested in the dialog box.

This topic explains how to set counterpoints on the sample code of the simulator also how to view the counterpoints.

To set the counterpoints in source code:

1. In the **CodeWarrior Projects** view, expand the **Source** folder under the project.
2. Double-click on the source file, for example, SC3900_main.c.

   The contents of the file appear on the right-side of the pane.
3. Select the line on which you want to set the counterpoint, and right-click on the Marker bar.

4. Select the **Toggle Counter Point** option from the context menu. The same option is used to remove the counterpoint from the Marker bar.

The counterpoint appears in blue color as shown in the figure.

5. Select the line on which you want to set another counterpoint. You can also set the counterpoints in the **Disassembly** view when the application is in debug mode.
6. After setting the counterpoints, debug the application and collect the data in the datafiles.

The datafile is generated after setting the counterpoints. If two different counterpoints are hit consecutively, the delta statistics will be computed between these two points as shown below.

![Figure 5-3. Second Counterpoint Set in Source Code](image)

![Figure 5-4. Events Collected After Setting Counterpoints](image)

You can match the counterpoint address with the trace address. A counterpoint is considered hit when its address is found in the collected trace.

The following table describes the fields of the counterpoints datafile.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Counter Point</td>
<td>The counterpoint that marks the beginning of the interval for which the delta statistics are computed</td>
</tr>
<tr>
<td>Destination Counter Point</td>
<td>The counterpoint that marks the end of the interval for which the delta statistics are computed</td>
</tr>
</tbody>
</table>

Table continues on the next page...
Table 5-1. Description of Fields - Counter Points Datafile (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chof Stalls</td>
<td>Displays the number of bubble cycles incurred by change of program flow for each instruction</td>
</tr>
<tr>
<td>Interlock Stalls</td>
<td>Stalls introduced by core to resolve resource conflicts</td>
</tr>
<tr>
<td>Memory Stalls</td>
<td>Displays the number of bubble cycles incurred by data bus hold for each instruction</td>
</tr>
<tr>
<td>Program Stalls</td>
<td>Displays the number of bubble cycles incurred by program starvation for each instruction</td>
</tr>
<tr>
<td>Rewind Stalls</td>
<td>Displays the number of bubble cycles incurred by a rewind on the core buses for each instruction</td>
</tr>
<tr>
<td>BTBable COF</td>
<td>The decoded COF is BTBable</td>
</tr>
<tr>
<td>COF Taken</td>
<td>The COF is taken; changes the program flow</td>
</tr>
<tr>
<td>False BTB Hit</td>
<td>Indicates False hit</td>
</tr>
</tbody>
</table>

NOTE
The mouse pointer over a counterpoint icon displays the attributes of counterpoints on that line.

Few examples are given below to understand how result is computed in different situations.

- If there is only source counterpoint with no destination, the counterpoint is hit only once, the statistics will show the events counted till that point.
Setting Counterpoints

Figure 5-6. Result Computed for Source Counterpoint Page

- If there is a counterpoint set in trace loop as shown in the below figure. The same counterpoint is found in trace consecutively and the delta statistics will be computed each time when that counterpoint is hit.

```c
sum_l0 = __l0_x(0);
sum_i3 = __l0_x(0);

cw_assert(length>5 & length%6==0);
for (i=0;i<length;i+=8)
{
    __ld_8l(px, sx0, sx1, sx2, sx3, sx4, sx5, sx6, sx7); px+=3;
    __ld_8l(py, sy0, sy1, sy2, sy3, sy4, sy5, sy6, sy7); py+=3;
    __macxml_cpp_2x(y0, y1, x0, x1, &sum_r0, &sum_10);    // x0 * y0 + x1 * y1
    __macxml_cpp_2x(y2, y3, x2, x3, &sum_r1, &sum_11);
    __macxml_cpp_2x(y4, y5, x4, x5, &sum_r2, &sum_12);
    __macxml_cpp_2x(y6, y7, x6, x7, &sum_r3, &sum_13);
}
__sod_all_2x(sum_r0, sum_i0, sum_r1, sum_i1, &sum_r0, &sum_i0);
```

Figure 5-7. Counterpoint Set after Loop Statement Page
5.2 Viewing Counterpoints in Analysispoints

You can view the attributes of counterpoints using Analysispoints view.

Analysis points view display the attributes of counterpoints set in source code or/and assembly code. To view the attributes of counterpoints, select Window > Show View > Other > Analysis > Analysispoints, the Analysispoints view is displayed in the below figure.

The source line counterpoints will have associated source file name and the line number, and the disassembly counterpoints will have associated source file name and the address.

You can also enable and disable counterpoints from this view. To enable or disable a particular counterpoint, right-click on the attribute and select Enable or Disable option from the menu.
5.3 Enabling and Disabling the Counterpoints

This section explains the procedure to enable or disable counterpoints.

If you want to enable the counterpoint, right-click on the Marker bar where counterpoint is already set and in disabled state, select the **Enable Counterpoint** option from the context menu.

If you want to disable the counterpoint, right-click on the Marker bar where counterpoint is already set and enabled, select the **Disable Counterpoint** option from the context menu. A disabled counterpoint will have no effect and will not compute the delta statistics. You can also disable the counterpoint from **Analysispoints** view. Right-click on the selected attribute and select **Disable** option. The unchecked attribute indicates the disabled counterpoint.

![Figure 5-10. Disable the Counterpoint Page](image-url)
Chapter 6
Performance Analysis for B4 Devices

The Performance Analysis tool enables you to configure, collect, and analyze scenarios in the B4 devices. The tool helps you optimize your bareboard application by using core and platform counters to provide enhanced levels of visibility to the application. This tool is integrated in the CodeWarrior IDE to handle configuration and communication with the target system.

**NOTE**

The Software Analysis and Performance Analysis tool cannot be used together as they use the same hardware resources. Currently, there is no mechanism for synchronization. You can either use Software Analysis or Performance Analysis.

This chapter explains:

- Performance Analysis Perspective
- Creating New Configuration
- Connecting to Board
- Selecting Scenarios
- Running Scenarios
- Viewing Results
- Custom scenarios

6.1 Performance Analysis Perspective

The **Performance Analysis** perspective provides a user-friendly interface to perform the scenario measurement activities, such as adding a configuration, configuring connection, defining scenarios, and collecting data.

The **Performance Analysis** perspective contains the following views:
6.1.1 Performance Analysis View

The **Performance Analysis** view organizes measurement activities in terms of projects. A project contains configuration information and the data that is collected as shown in the following figure.

![Performance Analysis View](image)

**Figure 6-1. Performance Analysis View**

You can import or export your project using the **Performance Analysis** view. To import a Performance Analysis configuration or project into your workspace, right-click the project and select the **Import** option from the context menu. In the **Import** wizard that appears, select the archived file or the root directory at which the project to be imported is located. Click **Finish**.
To export a Performance Analysis project, right-click the project and select the Export option from the context menu. In the Export wizard that appears, select the destination type and the directory where you want to export the project. Click Finish.

6.1.2 Configuration View

The Configuration view lets you configure connections and scenarios for measurement. The following figure shows the Configuration view.

![Figure 6-2. Configuration View](image)

The Configuration view contains the following panes:

- Connection Pane
- Scenarios Pane
- Sampling Pane
- Filter Core Events pane
6.1.2.1 Connection Pane

The **Connection** pane allows you to define the connection method that is used to communicate to the target hardware when a probe is being used in bareboard applications.

The following figure shows the **Connection Pane**.

![Connection Pane](image)

**Figure 6-3. Configuration View - Connection Pane**

The following table lists the options available in the **Connection** pane.

**Table 6-1. Configuration View - Connection Pane Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Dropdown" /></td>
<td>Drop-down menu</td>
<td>Lists the previously selected connections and displays the currently selected connection on top. Appears blank if there is no selected connection.</td>
</tr>
<tr>
<td><img src="image" alt="Plus" /></td>
<td>New Connection</td>
<td>Lets you add/configure a connection manually to a bareboard application. Refer <strong>Adding Connection Manually</strong> for more details.</td>
</tr>
<tr>
<td><img src="image" alt="Pencil" /></td>
<td>Edit Connection</td>
<td>Lets you modify an existing connection. The connection could be either automatically selected or manually added.</td>
</tr>
<tr>
<td><img src="image" alt="Minus" /></td>
<td>Remove Connection</td>
<td>Removes an existing connection from the list. Select the connection you want to remove and click this button.</td>
</tr>
<tr>
<td><img src="image" alt="Menu" /></td>
<td>Menu</td>
<td>Allows you to select an option to display the selected connection by IP address, host name or alias.</td>
</tr>
</tbody>
</table>

*Table continues on the next page...*
### Table 6-1. Configuration View - Connection Pane Options (continued)

<table>
<thead>
<tr>
<th>Option</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td></td>
<td>Displays the target processor. This determines what metrics and scenarios will be available to measure.</td>
</tr>
</tbody>
</table>

### 6.1.2.2 Scenarios Pane

The **Scenarios** pane lets you define the scenarios that will be measured.

A scenario contains the following:

- **Events** - Hardware occurrences that can be counted to any event in the B4 processor, including the output of other counters in a feedback loop.
- **Counters** - Counters that can be connected to any event in the B4 processor, including the output of other counters in a feedback loop.
- **Metrics** - A mathematical combination of counters (which count events) to provide an additional answer.

![Figure 6-4. Configuration View - Scenarios Pane](image)

**NOTE**

For B4 devices, 2 DDR Controllers are supported to measure DDR bandwidth with NDB (NXC Data Beats) counter.
The following table lists the options available in the **Scenarios** pane.

### Table 6-2. Configuration View - Scenarios Pane Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Icon]</td>
<td>Add a Stock Scenario</td>
<td>Lets you select different stock scenarios. A stock scenario is the standard scenario available on the tools. Refer <a href="#">Selecting Scenarios</a> for more details.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Delete a Stock Scenario</td>
<td>Removes the selected scenario from the list.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Move Scenarios Up</td>
<td>Moves scenarios up in the order of execution. This option is only available when multiple scenarios are added. The scenarios run in the order they appear in the Scenarios view. All scenarios are independent of each other.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Move Scenarios Down</td>
<td>Moves scenarios down in the order of execution. This option is only available when multiple scenarios are added. The scenarios run in the order they appear in the Scenarios view. All scenarios are independent of each other.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Show/Hide Advanced Options</td>
<td>When clicked to show advanced options, adds the <strong>Configure Counters</strong> column to the right of the <strong>Description</strong> column. The <strong>Configure Counters</strong> column displays the checkboxes against each scenario. If checked, the hardware is configured to measure the selected scenario when the profiling session starts. If unchecked, no hardware configuration happens, that is the previous hardware settings remain active and the current values of the counters are read.</td>
</tr>
</tbody>
</table>

### 6.1.2.3 Sampling Pane

The **Sampling** pane is used to configure the events sampling or collection settings.

The following figure shows an example of Sampling pane.
The table below lists the options available in the **Sampling** pane of the **Configuration** view.

### Table 6-3. Options Available in Sampling Pane

<table>
<thead>
<tr>
<th>Group</th>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Sampling Events</td>
<td></td>
<td>Lets you specify the settings for starting the event collection.</td>
</tr>
<tr>
<td></td>
<td>Automatically after launch</td>
<td>If selected, starts collecting events automatically when <strong>Profile</strong> button is clicked. This button is used to launch profiling to measure scenarios. For details, refer <a href="#">Running Scenarios</a>.</td>
</tr>
<tr>
<td></td>
<td>When I press the 'Go' toolbar button</td>
<td>If selected, waits for you to click the <strong>Go</strong> button after <strong>Profile</strong> button is clicked. The <strong>Go</strong> button is present in the <strong>Performance Analysis Session View</strong>.</td>
</tr>
<tr>
<td>Stop Sampling Events</td>
<td>After <code>&lt;number&gt;</code> samples</td>
<td>Stops event collection or scenario measurement after collecting a specified number of samples.</td>
</tr>
<tr>
<td></td>
<td>When I press the <code>Stop</code> toolbar button. Keep the last <code>&lt;number&gt;</code> samples</td>
<td>Stops event collection after you click the <strong>Stop</strong> button in the <strong>Performance Analysis Session View</strong>. The number of last collected samples that will be displayed need to be specified in the textbox.</td>
</tr>
<tr>
<td>Sample Values</td>
<td>Deltas</td>
<td>Lets you collect delta or absolute values from the target.</td>
</tr>
<tr>
<td></td>
<td>Absolute</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Halt Core Counters When Sampling</td>
<td></td>
</tr>
</tbody>
</table>

### 6.1.2.4 Filter Core Events pane
The **Filter Core Events** pane is used to limit the data collection between two addresses. For example, if you want the branch miss ration event generated only for a specific part of the code.

![Filter Core Events pane](image)

Figure 6-6. Configuration view - Filter Core Events pane

The table below lists the options available in the **Filter Core Events** pane of the **Configuration** view.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count events unconditionally</td>
<td>Collects events without specifying any range in the code.</td>
</tr>
<tr>
<td>Count events if PC is in Address 0..Address 1 Range</td>
<td>Counts events within a specified address range only.</td>
</tr>
<tr>
<td>Start counting at Address 0, stop at Address 1</td>
<td>Starts counting when address 0 is hit and stops at address 1.</td>
</tr>
<tr>
<td>Filter reference clock event</td>
<td>Allows you to filter the measurements on reference clock. This option becomes active only when you define a filter using any one of the above options, and if selected, applies that filter to the reference clock measurement.</td>
</tr>
</tbody>
</table>

### 6.1.2.5 Target Information Pane

The **Target Information** pane allows you to set the clock frequencies and multipliers on the target board.

These frequencies are used to compute complex metrics that imply reporting the number of events to a certain time frame, for example, computing the DDR traffic. The accuracy of the results depends on the correctness of the frequency values you specify.

The default values are displayed in the respective textboxes, you can modify them according to the speed on a particular target board.
6.1.2.6 Session pane

The Session pane lets you select the settings to be applied while running the scenario measurement session.

![Image of Session pane]

The table below lists the options available in the Session pane of the Configuration view.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatically display results</td>
<td>If selected, displays the generated data automatically after data measurement or when collection stops.</td>
</tr>
<tr>
<td>Stop session on first failure</td>
<td>If selected, stops collection of data when a failure is detected, for example, the application unable to configure some counters for measurement.</td>
</tr>
</tbody>
</table>

Table continues on the next page...
Table 6-5. Options available in Session pane (continued)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create log file with all target accesses</td>
<td>If selected, creates a log file which might be useful in case of a debug failure. You can view the log file in the Target Access Log tab, which appears (at the bottom of the results screen) after running a collection.</td>
</tr>
<tr>
<td>Enable CSV write during capture</td>
<td>If selected, allows you to save the data in CSV output file. You can select the CSV output directory using the Browse button.</td>
</tr>
<tr>
<td>Reset counters on launch</td>
<td>If selected, resets counters on the target when launching a new measurement session.</td>
</tr>
</tbody>
</table>

6.1.2.7 Realtime Visualization

The Realtime Visualization pane allows you to view the real time data of specified number of samples.

The following figure shown an example of Realtime Visualization pane.

![Realtime Visualization](image)

Figure 6-9. Configuration View - Realtime Visualization Pane

Check Enable Realtime Data Visualization checkbox to edit the number of samples to be displayed.

6.1.3 Performance Analysis Session View

The Performance Analysis Session view displays the status of all the events that are being generated when collection starts.

Once an event is generated successfully, the status is shown as Done. For pending events, the status is shown as Running.
The buttons available in the **Performance Analysis Session** view are:

- **Go** - Click this button to start data collection if you have selected the **When I press the `Go' toolbar button** option in the **Sampling Pane**, else the data collection starts automatically after pressing the **Profile** button.

- **Pause** - Click this button to pause data collection.
- **Stop** - Click this button to stop data collection.

### 6.1.4 Progress View
The Progress view of the Performance Analysis perspective displays the progress of the launching configuration.

The progress bar continues as events generate one by one. After complete generation, it displays the status as ‘No operations to display at this time’.

Figure 6-12. Progress View

6.2 Creating New Configuration

This section explains how to create a new configuration for Performance Analysis.

Perform the following steps to create a new configuration:

1. Select Window > Open Perspective > Other > Performance Analysis from the CodeWarrior IDE menu bar to open the Performance Analysis perspective.
2. Select File > New > Other to open the New wizard.
3. Select **Performance Analysis > Configuration** to open the **New Configuration** screen. Alternatively, click the icon from the main menu bar to get the **New Configuration** screen directly.
4. Type a name for the configuration in the **Configuration name** textbox.

5. Select **Bareboard Performance events** or **Linux application performance** depending upon the requirements.

6. Type a name for the project in the **Project name** textbox. This is the project in which your Performance Analysis configuration is defined. You can create the Performance Analysis configuration either in the existing project or in the new project. If you are creating the Performance Analysis project for the first time, the **Use existing project** option is disabled.

7. Click **Finish**.

The Performance Analysis project along with its configuration is created and appears in the **Performance Analysis** view.
6.3 Connecting to Board

You need to select a connection to the board after creating a Performance Analysis project.

You can select a connection to the board by manually configuring the connection.

- Adding Connection Manually

6.3.1 Adding Connection Manually

This section explains the steps required to manually configure the connection with the board.

Follow the steps to manually add a connection to the board:

1. Click the New Connection icon to open the Add a New Connection screen.
2. Select the processor and connection type in the respective fields.
3. Provide hostname or IP address depending on the connection type selected.
4. Specify an alias for the connection in the **Alias** textbox. This field is optional.
5. Click **OK** to save the settings.

The connection is set and selected in the **Configuration** view.
NOTE

If the connection displays unknown status, click the Scan Probe hyperlink to scan the probe connected to the processor.

6. Click File > Save to save your configuration.

6.4 Selecting Scenarios

This section explains the steps required to select and add a scenario to the Performance Analysis configuration.

Perform the following steps to select and add a scenario:

1. Click Add a Scenario in the Scenarios section of the Configuration view.

   ![Figure 6-17. Add a Scenario](image)

The Add <Processor> Scenarios screen appears in which only those scenarios that are applicable to the selected processor are displayed.

2. Select the subsystem that you want to analyze.
3. Click **OK** to add the selected scenarios to the configuration.

---

**Figure 6-18. Add <Processor> Scenarios Screen**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Core Related Scenarios</td>
</tr>
<tr>
<td><strong>Utilization</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Cache</strong></td>
<td></td>
</tr>
<tr>
<td>Data Cache Op</td>
<td>Measures Several Data Cache Operations</td>
</tr>
<tr>
<td>Data L1 Cache M</td>
<td>Measures Data L1 Cache Miss related statistics</td>
</tr>
<tr>
<td>Instruction L1 C</td>
<td>Measures Instructions L1 Cache miss related statistics</td>
</tr>
<tr>
<td>Backside L2 Cache Data M</td>
<td>Counts L2 Backside Cache Misses per L2 Data Cache Access</td>
</tr>
<tr>
<td>Backside L2 Cache Instruction C</td>
<td>Counts L2 Backside Cache Misses per L2 Instruction Cache Access</td>
</tr>
<tr>
<td><strong>MMU</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Core Complex</strong></td>
<td></td>
</tr>
<tr>
<td>SC3900</td>
<td>SC3900 Core Related Scenarios</td>
</tr>
<tr>
<td><strong>DDR</strong></td>
<td>DDR Related Scenarios</td>
</tr>
<tr>
<td><strong>DDR Traffic</strong></td>
<td>Measures whole system</td>
</tr>
<tr>
<td>DDR 1,2 Traffic w</td>
<td>DDR 1,2 statistics including read/write traffic, measured with hi...</td>
</tr>
<tr>
<td>DDR 1,2 Q0 Utili</td>
<td>DDR 1,2 q0 utilization</td>
</tr>
<tr>
<td><strong>CoreNet</strong></td>
<td>CoreNet Coherency Fabric Scenarios</td>
</tr>
<tr>
<td><strong>DPAA QMan</strong></td>
<td>Data Path Scenarios for the Queue Manager</td>
</tr>
<tr>
<td><strong>DPAA SEC</strong></td>
<td>Data Path Scenarios for the Security Engine</td>
</tr>
</tbody>
</table>
4. Click **File > Save** to save your configuration.

You can also modify your configuration settings such as configuring board connection or adding scenarios using the **Profile Configurations** dialog box. The **Profile Configurations** dialog box appears on selecting **Run > Profile Configurations** from the CodeWarrior IDE menu bar.

Select your configuration on left side, specify the required settings, and click **Apply** to save the settings.
6.5 Running Scenarios

This section explains about running the scenarios and viewing the result in the Performance Analysis view.

To run the selected scenarios, select Run > Profile from the CodeWarrior IDE menu bar to launch profiling. You can also click the Profile button in the main toolbar of the CodeWarrior IDE. Alternatively, you can click Profile in the Profile Configurations dialog box, which appears on selecting Run > Profile Configurations from the CodeWarrior IDE menu bar.

![Figure 6-21. Profiling/Measurement in Progress](image)

The Progress View provides a summary of the operations being carried out. Once the measurements are complete, the results are collected as part of the project and appear in the project folder in the Performance Analysis view.
6.6 Viewing Results

You can view the scenario measurement results either in tabular or graphical format.

To view the results, double-click the collections associated with your project in the **Performance Analysis** view. The generated data appears displaying a summary of the metrics and scenarios collected in a tabular format as shown in the following figure.

![Figure 6-22. Measurement Results](image)

**NOTE**

If you have kept the **Automatically display all results** checkbox checked in the **Session** pane of the **Configuration** view, the results will get open automatically while data is collecting.
At the bottom of the table view, there are four tabs:

- **Results** - Displays the default view showing the event data generated from the Performance Analysis tool. For details, refer Tabular Format.
- **Graphs** - Displays a graphical view of the data. For details, refer Graphical Format.
- **Configuration** - Displays a summary of the connection to the target.
- **Notes** - Allows you to write your own notes corresponding to the data results. The notes that you write get automatically saved which can be viewed later when you open the corresponding data results.
- **Delete** - Deletes the selected result.
- **Rename** - Lets you rename the selected result.
- **Import** - Lets you import the selected result into another project.
- **Export** - Lets you export results from another project.

### 6.6.1 Tabular Format
The tabular format of the results generated from the Performance Analysis tool displays the number of samples in rows and the number of events in columns.

There is one event per core generated in each column.

If you want to view event data of a particular column only, you can hide other columns. The columns hidden in the Results view are visible in the Graphs view.

Right-click a column header to view the various options available:

- **Export to CSV** - Allows you to export the selected column into a CSV file. Click this option to open the Export to CSV dialog box. Specify the location and name of the file and click Save.
- **Hide column** - Allows you to hide the selected column. To hide multiple columns, hold the control key and click through the columns.
- **Show all columns** - Displays all columns in the table.
- **Auto resize column** - Resizes all displayed columns according to table width.
- **Rename column** - Allows you to rename the selected column.

**6.6.2  Graphical Format**
The graphical format allows you to view the graphical representation of the results generated from the Performance Analysis tool.

Click the **Graphs** tab to view the graphical form of the collected data.

![Figure 6-25. Data Collection Results in Graphical Format](image)

A graph is generated for each event per core. You can use the scroll bar to view all the generated graphs for a particular collection. You can perform various tasks on the graphs, such as auto scaling, zooming-in/out, and panning. The table below lists the description of each option available on the graph.

**Table 6-6. Options Available in Graphical Format of Collected Results**

<table>
<thead>
<tr>
<th>Option</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Display Measurements</td>
<td>Allows you to specify strings to filter graphs according to their titles. For example, to display graphs with title BTB_MISS_RATIO (of Branch Miss Ratio event), specify the string BTB_MISS_RATIO in the text box, and click <strong>Apply</strong>. Only the graphs with</td>
</tr>
</tbody>
</table>

*Table continues on the next page...*
<table>
<thead>
<tr>
<th>Option</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BTB_MISS_RATIO as title will be displayed. If you want a specific string to be matched, uncheck the Regular Expression checkbox. If you want to match a particular string, anchor the string with * on both sides. For example, &quot;core6&quot; (with Regular Expressions unchecked) will display all results corresponding to core6 only. If you want to match a single character, anchor it with ? on both sides. For example, ?z?.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Configure Settings</td>
<td>Allows you to configure the settings of a selected graph. Click to display the Configure Graph Settings dialog box, and specify the graph settings (such as title color, legend, title font), axes settings (such as title, color, grid format), and traces settings (such as name, trace type, color, line width).</td>
</tr>
<tr>
<td></td>
<td>Add Annotation</td>
<td>Allows you to add annotations to a selected graph. Click to display the Add Annotation dialog box and specify the settings according to requirements.</td>
</tr>
<tr>
<td></td>
<td>Remove Annotation</td>
<td>Lets you remove the added annotations from a selected graph.</td>
</tr>
<tr>
<td></td>
<td>Perform Auto scale</td>
<td>Lets you adjust the x-axis and y-axis according to the range of the data in the selected graph.</td>
</tr>
<tr>
<td></td>
<td>Rubberband Zoom</td>
<td>Lets you select a rectangular area in the graph to zoom in.</td>
</tr>
<tr>
<td></td>
<td>Horizontal Zoom</td>
<td>Lets you specify the exact range horizontally in the graph to zoom in.</td>
</tr>
<tr>
<td></td>
<td>Vertical Zoom</td>
<td>Lets you specify the exact range vertically in the graph to zoom in.</td>
</tr>
<tr>
<td></td>
<td>Zoom In Horizontally</td>
<td>Zooms in horizontally in the area you clicked.</td>
</tr>
<tr>
<td></td>
<td>Zoom Out Horizontally</td>
<td>Zooms out horizontally in the area you clicked.</td>
</tr>
<tr>
<td></td>
<td>Zoom In Vertically</td>
<td>Zooms in vertically in the area you clicked.</td>
</tr>
<tr>
<td></td>
<td>Zoom Out Vertically</td>
<td>Zooms out vertically in the area you clicked.</td>
</tr>
<tr>
<td></td>
<td>Panning</td>
<td>Lets you move the selected graph to view a specific area of the graph.</td>
</tr>
<tr>
<td></td>
<td>Undo Panning</td>
<td>Lets you undo the panning action on the graph.</td>
</tr>
<tr>
<td></td>
<td>Redo Panning</td>
<td>Lets you redo the panning action on the graph.</td>
</tr>
</tbody>
</table>
### 6.7 Custom scenarios

The Performance Analysis tool allows you to select predefined analysis configurations called **Stock Scenarios**.

A stock scenario configures events that are to be measured and predefined metrics to be calculated on the collected data.

As you become more familiar with the system that you are measuring, the ability to create your own scenarios becomes crucial. The Performance Analysis tool lets you create a custom scenario in order to minutely narrow or extend the analysis.

This topic contains the following sub-topics.

- Defining custom scenarios
- Editing custom scenarios

#### 6.7.1 Defining custom scenarios

This section explains the steps to define a custom scenario.

Perform the following steps to define a custom scenario:

1. Choose **File > New > Other** to open the **New** wizard.
2. Choose **Performance Analysis > Custom Scenario** and click **Next**.
   
   The **New Custom Scenario** screen appears.

3. Specify the **Processor**, **Name**, and **Description** for the scenario in the respective fields.
4. Click **Next** and select an existing scenario to be used as a starting point. By default, scenarios are grouped by system component; you can choose to group the scenarios by measurement type using the **Group By** pop-up menu.
5. Click **Finish** to complete the creation of a custom scenario.
The New wizard closes.

6.7.2 Editing custom scenarios

After defining a custom scenario using the New wizard, a node appears for Custom Scenarios in the Performance Analysis view. Expand this node and select the custom scenario. Double-click the scenario to open the Custom Scenario Editor. The Custom Scenario Editor allows you to modify the custom scenarios that you have defined according to your requirements.

You can add events, counters and metrics to the custom scenario using Custom Scenario Editor.

To modify a custom scenario, use the following tabs of the Custom Scenario Editor.

- Overview
- Events
- Metrics
- Initialization
- Reports

6.7.2.1 Overview

The Overview tab of the Custom Scenario Editor provides a brief summary of the scenario you are editing.

It replicates the information that was entered during the creation of the custom scenario. You can edit the processor to be used for the scenario, provide an alias name to the scenario, or modify the description of the custom scenario using the respective fields.
6.7.2.2 Events

The **Events** tab of the **Custom Scenario Editor** lists all of the device's components that have events available for configuration, in the **Available Events** pane.

There can be hundred of these events, so to make it easier to locate an event of interest, this tab provides a search text box where you can enter a search string. For example, in the figure below, as the word *thread* is entered, the tool has displayed all events that include the word *thread* in their description.

Once you locate an event of interest, select it and click the **Add** button. The selected event will be added to the list of **Selected Events** pane on the right side. The Performance Analysis tool automatically handles the allocation of the event to a given counter and manages the available counters as well. As soon as all counters have been assigned, any other events listed will be grayed out. If you try to add an event from the grayed out list, the tool displays an error message indicating that there are no more counters available to assign.
6.7.2.3 Metrics

The Metrics tab of the Custom Scenario Editor lets you define (optionally) metrics (mathematical operations) of the measured events.

The Metrics tab provides a toolbar, which allows you:

- Add a metric
- Edit an existing metric that is selected in the table
- Remove an existing metric that is selected in the table
To add a metric to a custom scenario, click the **Add Metric** button in the toolbar to open the **New Metric** dialog. The **Add Metric** button provides a pop-up menu containing two options: **Add New Metric** and **Add Stock Metrics**.
Specify the Name, Expression, andQualifier for the metric using the respective fields. The Expression represents a combination of event names and mathematical operands. The Events button allows you to select the necessary events for a metric. You can also select a constant to be used within the metric expression using the Constants button. As metrics are operations on events, you will also need to choose an appropriate operator. Click the Operators button to view the list of C language operator supported on the metric expressions. Click OK to complete the definition of the metric. The Qualifier represents a tag or a label. It helps to define filters based on an attribute/property.

Click the Add Stock Metrics option to open the Stock Metrics dialog, which allows you to select a metric from a list of predefined metrics to be added to a custom scenario. You can add a stock metric by selecting the checkbox corresponding to the metric. To enable a stock metric, right-click it and choose the Add Events to Enable Metric option.

Some of the metrics in the Stock Metrics dialog are disabled due to missing events. The Performance Analysis tool automatically handles the allocation of the event to a given counter and manages the available counters as well. As soon as all counters have been assigned, any other events listed will be grayed out. If you try to add an event from the grayed out list, the tool displays an error message indicating that there are no more counters available to assign.

### 6.7.2.4 Initialization

Use the Initialization tab in situations where you may require advanced target initialization beyond the configuration that enables the event-counter pairs.

The Initialization tab of the Custom Scenario Editor allows you to enter a python code or specify a python script directly. Select the Enter Python Code option to directly write the python code in the available text editor. Select the Specify Python Script File option to search the workspace or file system for previously saved python scripts.
6.7.2.5 Reports

The **Reports** tab of the **Custom Scenario Editor** displays the reports of the event and metric measurements that you have added to the custom scenario.
The toolbar next to the **Reports** pane provides the following buttons.
- **New Report** - Click to open the **New Report** dialog and create a new report. Specify the report **Type**, **Name**, and **Description** in the respective fields. Select the **Automatically add new events and metrics** checkbox to automatically include in the report the new measurements that you add to the custom scenario.
- **Edit Report** - Click to open the **Edit Report** dialog to modify the selected report.
- **Delete Report** - Click to delete the selected report.
- **Move Report Up** - Click to move the selected report up in the list.
- **Move Report Down** - Click to move the selected report down in the list.

The toolbar next to the **Measurements** pane provides the following buttons.
- **Add Measurement** - Click to open the **Add Measurement** dialog. This dialog lets you select the events and metrics that you want to add to the report. This dialog displays only those events and metrics that have not been added yet to the report.
- **Remove Measurement** - Click to delete the selected measurement.
- **Move Measurement Up** - Click to move the selected measurement up in the list.
- **Move Measurement Down** - Click to move the selected measurement down in the list.
Chapter 7
Launching Scripts

The Profiling and Analysis tools help you launch scripts written in Python language. These scripts allow you to collect and export trace data automatically using remote launch. This feature is integrated with CodeWarrior IDE scripting. The scripts are launched in remote launch using the Jython console. The remote launch feature of CodeWarrior allows launch configurations to be executed remotely. For more details on API of remote launch, refer to the Remote Launch API Appendix.

In this chapter:

- Run Sample Python Script
- Collect Trace Using Jython

### 7.1 Run Sample Python Script

The steps in this topic demonstrate the procedure of running a sample Python script from Jython console, that is running the script that you might have written for collecting trace data.

Perform the following steps:

1. Create a new B4860 project with the name remote-launch.
2. Replace the source code of b4860_main.c with your own code.
3. Save and build your project.
4. Update the project name, launch config name and start/stop tracepoints line numbers in the TestHwTracepointsProgTrace.py script according to your project that is created, TestHwTracepointsProgTrace.py file is located at: \<CW installation directory>\eclipse \plugins\com.freescale.sa.sc.scripting_*\remote\scripting_sa_sc.
5. Select Window > Show View > Other > Debug > Remote Launch to open the Remote Launch view.
6. Click the **Enable Remote Launch** button displayed on the top right of the view to enable remote launching.

   The color of the button changes to red.

7. Select **Window > Show View > Other > Debug > Jython Consoles** to open the **Jython Consoles** view.

8. Open the **View** menu in the **Jython Consoles** view, and select the **New Interpreter** option.

![Input Name](image)

**Figure 7-1. Select New Interpreter Option in Jython Consoles View**

The **Input Name** dialog box appears.

9. Specify a name for the new interpreter and click **OK**.

10. In the Jython console command prompt (">>>"), input code in Python syntax. You may access any classes in your test scripts, for example, `TestHwTracepointsProgTrace.py`, by inputting the following commands in the Jython command prompt:

    ```python
    from scripting_sa_sc import TestHwTracepointsProgTrace
    RemoteLaunchUtils.executeLaunchScript(TestHwTracepointsProgTrace())
    ```

**NOTE**

The advantage of using Jython is that besides Python modules, it can also access Java modules in the similar manner.

This is how you run a sample Python script from Jython console.

### 7.2 Collect Trace Using Jython

This section demonstrates the procedure of launching configuration and collecting both manual and automatic trace on the B4860 target.
To collect trace using remote launch through Jython console, perform the following steps.

**Figure 7-2. Running Commands at Jython Command Prompt for Collecting Trace**

The trace collection starts and the Software Analysis view appears with the trace results. Open the Software Analysis view and check the collected results.

**Figure 7-3. Software Analysis View Containing Trace Results Collected Using Remote Launch**

Click Trace and check the collected results.
**Figure 7-4. Trace Results Collected Using Remote Launch**

**NOTE**

The sample scripts used for collecting continuous and automatic trace on the B4860 targets are located at: `<CW\Installation Directory>\eclipse\plugins\com.freescale.sa.sc.scripting_*\remote\scripting_sa_sc.`

This is how you collect trace using the Jython console.
Chapter 8
Remote Launch API

This appendix documents following classes and interface for the remote launch API.

Classes
- com::freescale::sa::scripting::api::AnalysisActions
- com::freescale::sa::scripting::api::AnalysisConfig
- com::freescale::sa::scripting::api::AnalysisErrors
- com::freescale::sa::scripting::api::AnalysisFactory
- com::freescale::sa::scripting::api::AnalysisResults
- com::freescale::sa::sc::scripting::api::ScAnalysisConfig
- com::freescale::sa::sc::scripting::api::ScHwAnalysisActions
- com::freescale::sa::sc::scripting::api::ScHwAnalysisConfig
- com::freescale::sa::sc::scripting::api::ScSimAnalysisConfig
- com::freescale::sa::scripting::api::TracePoint

Interface
- com.freescale.sa.scripting.IAnalysisConfigFactory

8.1 com::freescale::sa::scripting::api::AnalysisActions

Refer below.

8.1.1 Public Member Functions

Lists the public member functions of the com::freescale::sa::scripting::api::AnalysisActions class.

Following public members are available:
- AnalysisActions
Comprehensive description of each of the above listed functions is as follows:

### 8.1.1.1 AnalysisActions

**Constructor.**

```java
```

**Parameters**

`config`

The AnalysisConfig-inherited instance, which carries the information needed to identify the running launch.

### 8.1.1.2 startTrace

Start trace collection.

```java
synchronized void com.freescale.sa.scripting.api.AnalysisActions.startTrace()
```

### 8.1.1.3 stopTrace

Stop trace collection.

```java
synchronized void com.freescale.sa.scripting.api.AnalysisActions.stopTrace()
```

### 8.1.2 Public Static Member Functions

List the public static member functions of the `com::freescale::sa::scripting::api::AnalysisActions` class.

Following public static members are available:
• startTrace
• stopTrace

Comprehensive description of each of the above listed functions is as follows:

8.1.2.1 startTrace

Starts trace collection.

This method is intended for a static use, where an analysis configuration is not available.

```java
static synchronized void com.freescale.sa.scripting.api.AnalysisActions.startTrace(String launchConfigName)
```

**Parameters**

`launchConfigName`

The name of the running launch configuration, on which the action is triggered.

8.1.2.2 stopTrace

Stop trace collection.

This method is intended for a static use, where an analysis configuration is not available.

```java
static synchronized void com.freescale.sa.scripting.api.AnalysisActions.stopTrace(String launchConfigName)
```

**Parameters**

`launchConfigName`

The name of the running launch configuration, on which the action is triggered.

8.2 com::freescale::sa::scripting::api::AnalysisConfig

Refer below.

8.2.1 Protected Member Functions
Lists the protected member functions of the `com::freescale::sa::scripting::api::AnalysisConfig` class.

Following protected members are available:

- `AnalysisConfig`
- `setProject`
- `load`
- `save`
- `getWorkingCopy`
- `getLaunchConfiguration`

Comprehensive description of each of the above listed functions is as follows:

### 8.2.1.1 AnalysisConfig

`com.freescale.sa.scripting.api.AnalysisConfig.AnalysisConfig(String projectName)`

### 8.2.1.2 setProject

`void com.freescale.sa.scripting.api.AnalysisConfig.setProject(ILaunchConfiguration launchConfig)`

### 8.2.1.3 load

`void com.freescale.sa.scripting.api.AnalysisConfig.load()`

### 8.2.1.4 save

Saves current configuration.

`void com.freescale.sa.scripting.api.AnalysisConfig.save()`

### 8.2.1.5 getWorkingCopy
Get the working copy of the active launch configuration.

ILaunchConfigurationWorkingCopy
com.freescale.sa.scripting.api.AnalysisConfig.getWorkingCopy()

**Returns**
The current launch configuration working copy.

### 8.2.1.6 getLaunchConfiguration

ILaunchConfiguration com.freescale.sa.scripting.api.AnalysisConfig.getLaunchConfiguration()

### 8.2.2 Public Member Functions

Lists the public member functions of the
com::freescale::sa::scripting::api::AnalysisConfig class.

Following public member functions are available:

- getOutputFolder
- addSrcLineTracePoint
- addAddressTracePoint
- removeTracePoint
- getTracePoints
- setIgnoreAllTracePoints
- getIgnoreAllTracePoints
- removeAllTracePoints
- removeLineCounterpoint
- removeAddrCounterpoint
- setLineCounterpointEnabled
- setAddrCounterpointEnabled
- getAttribute
- setAttribute

### 8.2.2.1 getOutputFolder

Returns the SA output folder attribute.
String com.freescale.sa.scripting.api.AnalysisConfig.getOutputFolder()

**Returns**

The output folder path.

### 8.2.2.2 addSrcLineTracePoint

Adds a tracepoint identified by a source file and a line number to a project.

```
TracePoint com.freescale.sa.scripting.api.AnalysisConfig.addSrcLineTracePoint(String sourceFilePath, int lineNo, TracePoint.Action action, TracePoint.Type type)
```

**Parameters**

- **sourceFilePath**
  
  Path towards source file of tracepoint.

- **lineNo**
  
  Line Number of tracepoint.

- **action**
  
  Action of tracepoint, see for possible values. TracePoint.Action

- **type**
  
  Type of tracepoint, see for possible values. TracePoint.Action

**Returns**

Return the resulted TracePoint

### 8.2.2.3 addAddressTracePoint

Adds an tracepoint identified by an address to a project.

```
TracePoint com.freescale.sa.scripting.api.AnalysisConfig.addAddressTracePoint(String sourceFilePath, String address, TracePoint.Action action, TracePoint.Type type, int lineNo)
```

**Parameters**

- **sourceFilePath**
  
  Path towards source file where marker for tracepoint should be displayed. Set it to empty string and no marker will be displayed for this tracepoint in source file.
address
Address of tracepoint.

action
Action of tracepoint, see for possible values. TracePoint.Action

type
Type of tracepoint, see for possible values. TracePoint.Action

lineNo
Line number from source file where marker for tracepoint should be displayed. Set it to 0 and no marker will be displayed for this tracepoint in source file.

coreException

Returns
Return the resulted . TracePoint

8.2.2.4 removeTracePoint

Remove tp tracepoint from the current project.

int com.freescale.sa.scripting.api.AnalysisConfig.removeTracePoint(TracePoint tp)

Parameters

tp
Tracepoint to be removed.

Returns
AnalysisErrors.STATUS_OK if the operation was successful, or AnalysisErrors.STATUS_ERROR if it fails.

8.2.2.5 getTracePoints

Get all tracepoints from the current project.

Vector com.freescale.sa.scripting.api.AnalysisConfig.getTracePoints()
All tracepoints from the current project

8.2.2.6 setIgnoreAllTracePoints

Set all tracepoints to be ignored or not.

```java
void com.freescale.sa.scripting.api.AnalysisConfig.setIgnoreAllTracePoints(boolean ignoreAll)
```

**Parameters**

- `ignoreAll`
  
  True for ignore all tracepoints, false otherwise.

8.2.2.7 getIgnoreAllTracePoints

Get ignore all tracepoints status.

```java
boolean com.freescale.sa.scripting.api.AnalysisConfig.getIgnoreAllTracePoints()
```

**Returns**

True if all tracepoints are currently ignored, false otherwise.

8.2.2.8 removeAllTracePoints

Remove all tracepoints from the current project.

```java
int com.freescale.sa.scripting.api.AnalysisConfig.removeAllTracePoints()
```

**Returns**

- `AnalysisErrors.STATUS_OK` if the operation was successful, or
- `AnalysisErrors.STATUS_ERROR` if it fails.

8.2.2.9 removeLineCounterpoint

Remove a counterpoint from a C source file line number.
void com.freescale.sa.scripting.api.AnalysisConfig.removeLineCounterpoint(int lineNumber, String sourceFilePath)

**Parameters**

**lineNumber**

The line number the counterpoint to be added to.

**sourceFilePath**

The path to the source file, where the counterpoint will be added.

CoreException

### 8.2.2.10 removeAddrCounterpoint

Remove an address counterpoint.

void com.freescale.sa.scripting.api.AnalysisConfig.removeAddrCounterpoint(String hexAddr)

**Parameters**

**hexAddr**

The hex address at which this counterpoint is set.

CoreException

### 8.2.2.11 setLineCounterpointEnabled

Enable or disable a line counterpoint.

void com.freescale.sa.scripting.api.AnalysisConfig.setLineCounterpointEnabled(int lineNumber, String sourceFilePath, boolean enabled)

**Parameters**

**lineNumber**

The line number the counterpoint needs to be set up at.

**sourceFilePath**

The path to the C source file.

**enabled**
Set counterpoint enabled or disabled.

CoreException

8.2.2.12  `setAddrCounterpointEnabled`

Enable or disable an address counterpoint.

```java
void com.freescale.sa.scripting.api.AnalysisConfig.setAddrCounterpointEnabled(boolean enabled, String hexAddr)
```

**Parameters**

- `enabled`
  
  Set counterpoint enabled or disabled.

- `hexAddr`
  
  The HEX address the counterpoint needs to be set up at.

CoreException

8.2.2.13  `getAttribute`

Get the value of a SA general configuration attribute.

```java
Object com.freescale.sa.scripting.api.AnalysisConfig.getAttribute(Object attributeType)
```

**Parameters**

- `attributeType`
  
  The attribute from which the value will be retrieved. The attributes are exposed by the `AnalysisConfigAttributes` enum.

**Returns**

The specific attribute value.

8.2.2.14  `setAttribute`

Set one of the SA general config attributes.
void com.freescale.sa.scripting.api.AnalysisConfig.setAttribute(Object attributeType, Object arg)

Parameters

attributeType

One of the AnalysisConfigAttributes enum attributes (SET_AUTO_TRACE_COLLECTION).

arg

The value the attribute will take.

8.2.3 Public Static Member Functions

Lists the public static member functions of the com::freescale::sa::scripting::api::AnalysisConfig class.

Following public static member functions are available:

• removeAllTracepointsByType

8.2.3.1 removeAllTracepointsByType

Remove all tracepoints of the same kind.

static void com.freescale.sa.scripting.api.AnalysisConfig.removeAllTracepointsByType(TracePoint.Type tpType)

Parameters

tpType

Tracepoint type.

CoreException

8.3 com::freescale::sa::scripting::api::AnalysisErrors

Refer below.
8.3.1 Public Static Member Functions

Lists the public static member functions of the \texttt{com::freescale::sa::scripting::api::AnalysisErrors} class.

Following public static member functions are available:

- \texttt{log}

8.3.1.1 \texttt{log}

Log error.

\texttt{static void com.freescale.sa.scripting.api.AnalysisErrors.log( Throwable e )}

**Parameters**

- \texttt{e}

  A throwable object.

8.4 \texttt{com::freescale::sa::scripting::api::AnalysisFactory}

Refer below.

8.4.1 Public Static Member Functions

Lists the public static member functions of the \texttt{com::freescale::sa::scripting::api::AnalysisFactory} class.

Following public static members are available:

- \texttt{getAnalysisConfig}
- \texttt{getAnalysisConfigFromLaunch}
- \texttt{getAnalysisResults}
- \texttt{getAnalysisResults}
8.4.1.1  getAnalysisConfig

Get which configures the SA configuration attributes. AnalysisConfig

```java
static AnalysisConfig
com.freescale.sa.scripting.api.AnalysisFactory.getAnalysisConfig(String projectName)
```

**Parameters**

`projectName`

The project name.

**Returns**

An object. AnalysisConfig

8.4.1.2  getAnalysisConfigFromLaunch

Get which configures the SA configuration attributes. AnalysisConfig

This method has a more focused use case, where the user has more than one launch configuration and he needs to specifically select one for the launch script.

```java
static AnalysisConfig
com.freescale.sa.scripting.api.AnalysisFactory.getAnalysisConfigFromLaunch(String launchConfigName)
```

**Parameters**

`launchConfigName`

The launch configuration name, as it is described only by the base name without the file's extension.

**Returns**

An object. AnalysisConfig

8.4.1.3  getAnalysisResults
Get which helps in exporting trace information. `AnalysisResults`

```java
static AnalysisResults
com.freescale.sa.scripting.api.AnalysisFactory.getAnalysisResults(String projectName, int timeout)
```

### Parameters

**projectName**

The name of the project.

**timeout**

Maximum allowed time for the results to become available.

### Returns

An instance. `AnalysisResults`

---

### 8.4.1.4 `getAnalysisResults`

Get which helps in exporting trace information. `AnalysisResults`

This method returns an `AnalysisResult` instance with a default 10 second wait for results availability.

```java
static AnalysisResults
com.freescale.sa.scripting.api.AnalysisFactory.getAnalysisResults(String projectName)
```

### Parameters

**projectName**

The name of the project.

### Returns

An instance. `AnalysisResults`

---

### 8.4.1.5 `getAnalysisResults`

Get which helps in exporting trace information. `AnalysisResults`

```java
static AnalysisResults
com.freescale.sa.scripting.api.AnalysisFactory.getAnalysisResults(String projectName, String baseFileName, int timeout)
```
Parameters

projectName
The name of the project.

baseFileName
Base name, without extension, of the result set
timeout
Base name, without extension, of the result set

Returns
An instance. AnalysisResults

8.4.1.6 getAnalysisResults

Get which helps in exporting trace information. AnalysisResults

This method returns an AnalysisResult instance with a default 10 second wait for results availability.

static AnalysisResults
com.freescale.sa.scripting.api.AnalysisFactory.getAnalysisResults(String projectName, String baseFileName)

Parameters

projectName
The name of the project.

baseFileName
Base name, without extension, of the result set

Returns
An instance. AnalysisResults

8.5 com::freescale::sa::scripting::api::AnalysisResults

Refer below.
8.5.1 Protected Member Functions

Lists the protected member functions of the `com::freescale::sa::scripting::api::AnalysisResults` class.

Following protected members are available:

- `AnalysisResults`
- `AnalysisResults`
- `exportTraceData`

Comprehensive description of each of the above listed functions is as follows:

8.5.1.1 AnalysisResults

```java
com.freescale.sa.scripting.api.AnalysisResults.AnalysisResults(String projectName, String baseName, int timeoutSeconds)
```

**Parameters**

`projectName`

The name of the current project.

`baseName`

Base name, without extension, of the result set

`timeoutSeconds`

Maximum allowed time for the results to become available, in seconds.

8.5.1.2 AnalysisResults

Overloaded constructor which uses the default waiting time for the target file (i.e. config file) file.

```java
com.freescale.sa.scripting.api.AnalysisResults.AnalysisResults(String projectName, String baseFileName)
```

**Parameters**
The name of the current project.

Base name, without extension, of the result set

### 8.5.1.3 exportTraceData

```java
void com.freescale.sa.scripting.api.AnalysisResults.exportTraceData(String filePath)
```

### 8.5.2 Public Member Functions

Lists the public member functions of the `com::freescale::sa::scripting::api::AnalysisResults` class.

Following public members are available:

- `getTraceRecordsNo`
- `exportResults`
- `exportFunctionResults`

Comprehensive description of each of the above listed functions is as follows:

#### 8.5.2.1 getTraceRecordsNo

```java
long com.freescale.sa.scripting.api.AnalysisResults.getTraceRecordsNo()
```

#### 8.5.2.2 exportResults

Export to a data format one of the available data types.

The data types that can be exported with this method can take only one argument.

```java
com.freescale.sa.scripting.api.AnalysisResults.exportResults(ExportDataType dataType, String filePath)
```

**Parameters**
8.5.2.3 exportFunctionResults

Export to a data format one of the available data types.

The data types that can be exported with this method should take two arguments.

```
com.freescale.sa.scripting.api.AnalysisResults.exportFunctionResults(ExportDataType
dataType, String filePath, Object arg)
```

**Parameters**

**dataType**

the data type as one of the attributes from ExportDataType enum.

**filePath**

the path to the desired .csv output file.

**arg**

Defines the SHOW_CODE_TYPE for the CRITICAL_CODE_HTML attribute or the name of the analysed C function (one of the source file's functions) for CRITICAL_CODE_FUNCTION_CSV and HIERARCHICAL_FUNCTION_CSV.

8.5.3 Public Static Member Functions

Lists the public static member functions of the
```
com.freescale.sa.scripting.api.AnalysisResults
```
class.

Following public static members are available:

- logMessages

Comprehensive description of each of the above listed functions is as follows:
8.5.3.1 logMessages

Method used for temporary logging.
Afterwards, the logging will be done by the RemoteLaunch API.

```java
static void com.freescale.sa.scripting.api.AnalysisResults.logMessages(String message)
```

**Parameters**

- `message`
The message to be logged.

8.6 com::freescale::sa::sc::scripting::api::ScAnalysisConfig

Refer below.

8.6.1 Protected Member Functions

Lists the protected member functions of the `com::freescale::sa::sc::scripting::api::ScAnalysisConfig` class.

Following protected members are available:

- `ScAnalysisConfig`
- `ScAnalysisConfig`

8.6.1.1 ScAnalysisConfig

```java
com.freescale.sa.sc.scripting.api.ScAnalysisConfig.ScAnalysisConfig(ILaunchConfiguration launchConfig)
```

8.6.1.2 ScAnalysisConfig
8.6.2 Public Member Functions

Lists the public member functions of the 
com::freescale::sa::sc::scripting::api::ScAnalysisConfig class.

Following public members are available:

- `getAttribute`
- `setAttribute`

Comprehensive description of each of the above listed functions is as follows:

8.6.2.1 `getAttribute`

Get the value of a specific SA StarCore configuration attribute.

```java
Object com.freescale.sa.sc.scripting.api.ScAnalysisConfig.getAttribute(Object attributeType)
```

**Parameters**

attributeType

The attribute from which the value will be retrieved. The list of attributes are exposed by the ScAnalysisConfigAttributes enum.

**Returns**

The specific attribute value or null in case of attribute not found.

8.6.2.2 `setAttribute`

Set one of the SA StarCore general config attributes.

```java
void com.freescale.sa.sc.scripting.api.ScAnalysisConfig.setAttribute(Object attributeType, Object arg)
```

**Parameters**

attributeType
One of the AnalysisConfigAttributes enum attributes.

arg

The value the attribute will take.

8.7  com::freescale::sa::sc::scripting::api::ScHwAnalysisActions

Refer below.

8.7.1  Public Member Functions

Lists the public member functions of the
com::freescale::sa::sc::scripting::api::ScHwAnalysisActions class.

Following public member functions are available:

- ScHwAnalysisActions
- uploadTrace

Comprehensive description of each of the above listed functions is as follows:

8.7.1.1  ScHwAnalysisActions

It enables user defined events trace.

com.freescale.sa.sc.scripting.api.ScHwAnalysisActions.ScHwAnalysisActions(AnalysisConfig config)

8.7.1.2  uploadTrace

Trigger trace upload.

This should be triggered only in attach mode or user code, when a suspend is hit, after tracing was enabled.

synchronized void com.freescale.sa.sc.scripting.api.ScHwAnalysisActions.uploadTrace()
8.7.2 Public Static Member Functions

Lists the public static member functions of the
\texttt{com::freescale::sa::sc::scripting::api::ScHwAnalysisActions} class.

Following public static members are available:

- \texttt{uploadTrace}

Comprehensive description of each of the above listed functions is as follows:

8.7.2.1 uploadTrace

This method is intended for a static use, where an analysis configuration is not available. This should be triggered only in attach mode or user code, when a suspend is hit, after tracing was enabled.

\begin{verbatim}
static synchronized void com.freescale.sa.sc.scripting.api.ScHwAnalysisActions.uploadTrace(String launchConfigName)
\end{verbatim}

\textbf{Parameters}

- \texttt{launchConfigName}

The name of the running launch configuration, on which the action is triggered.

8.8 \texttt{com::freescale::sa::sc::scripting::api::ScHwAnalysisConfig}

Refer below.

8.8.1 Public Member Functions

Lists the public member functions of the
\texttt{com::freescale::sa::sc::scripting::api::ScHwAnalysisConfig} class.

Following public member functions are available:

- \texttt{ScHwAnalysisConfig}
• ScHwAnalysisConfig
  • setAttribute
  • getAttribute

8.8.1.1 ScHwAnalysisConfig

com.freescale.sa.sc.scripting.api.ScHwAnalysisConfig.ScHwAnalysisConfig(ILaunchConfiguration launchConfig)

8.8.1.2 ScHwAnalysisConfig

com.freescale.sa.sc.scripting.api.ScHwAnalysisConfig.ScHwAnalysisConfig(String projectName)

8.8.1.3 setAttribute

Set one of the SA StarCore hardware config attributes. If called from python interpreter, there's no need to mention the enum name (e.g. ScHwAnalysisConfigAttributes.USER_CODE, just USER_CODE). The enum objects are automatically added as they are to the namespace of the running project, via a config.py file, found in com.freescale.sa.sc.scripting/remote.

void com.freescale.sa.sc.scripting.api.ScHwAnalysisConfig.setAttribute(Object attributeType, Object arg)

Parameters

attributeType

One of the ScHwAnalysisConfigAttributes enum attributes (TRACE_SCENARIO, OWNERSHIP_TRACE, USER_DEFINED_EVENTS, TRACE_BUFFER_START_ADDRESS, DATA_ADDR_TRACE, PROBE_BUFFER_SIZE, TRIAD_A_EVENT, TRIAD_B_EVENT, TRACE_COLLECTION_MODE, USER_CODE, DDR_BUFFER_START_ADDRESS, DDR_BUFFER_SIZE, SET_LCF_SETTINGS_FROM_FILE, BANDWIDTH, COLLECTION_PORT_TYPE).

Parameters

arg
The value the attribute will take. In case, the attribute to be set is ambiguous, consult com.freescale.sa.sc.scripting/remote/config.py, where the scripting namespace is set. There can be found possible values for multiple-choice arguments(e.g. setTriadAEvent(..) or setTraceCollectionMode(..) ).

### 8.8.1.4 getAttribute

Get the value of a specific SA StarCore hardware config attribute. If called from python interpreter, there's no need to mention the enum name (e.g. ScHwAnalysisConfigAttributes.USER_CODE, just USER_CODE). The enum objects are automatically added as they are to the namespace of the running project, via a config.py file, found in com.freescale.sa.sc.scripting/remote.

Object com.freescale.sa.sc.scripting.api.ScHwAnalysisConfig.getAttribute (Object attributeType)

**Parameters**

attributeType

One of the ScHwAnalysisConfigAttributes enum attributes (TRACE_SCENARIO, OWNERSHIP_TRACE, USER_DEFINED_EVENTS, TRACE_BUFFER_START_ADDRESS, DATA_ADDR_TRACE, PROBE_BUFFER_SIZE, TRIAD_A_EVENT, TRIAD_B_EVENT, TRACE_COLLECTION_MODE, USER_CODE, DDR_BUFFER_START_ADDRESS, DDR_BUFFER_SIZE, SET_LCF_SETTINGS_FROM_FILE, BANDWIDTH, COLLECTION_PORT_TYPE).

**Returns**

the specific attribute value.

### 8.8.2 Protected Member Functions

Lists the protected member functions of the com.freescale.sa.sc.scripting.api:ScHwAnalysisConfig class.

Following protected members are available:

- setUserDefinedEventsTrace
- isUserDefinedEventsTrace
- setOwnershipTrace
• isOwnershipTrace
• setTraceScenario
• getTraceScenario
• load

Comprehensive description of each of the above listed functions is as follows:

### 8.8.2.1 setUserDefinedEventsTrace

It enables user defined events trace.

```java
void com.freescale.sa.sc.scripting.api.ScHwAnalysisConfig.setUserDefinedEventsTrace(boolean userDefinedEvents)
```

**Parameters**

- `userDefinedEvents`

Option to enable or not user defined events trace.

### 8.8.2.2 isUserDefinedEventsTrace

It returns the enable state of user defined events trace.

```java
boolean com.freescale.sa.sc.scripting.api.ScHwAnalysisConfig.isUserDefinedEventsTrace()
```

### 8.8.2.3 setOwnershipTrace

It enables ownership trace.

```java
void com.freescale.sa.sc.scripting.api.ScHwAnalysisConfig.setOwnershipTrace(boolean ownershipTrace)
```

**Parameters**

- `ownershipTrace`

Option to enable or not ownership trace.
8.8.2.4 isOwnershipTrace

It returns the enable state of ownership trace.

\[
\text{boolean com.freescale.sa.sc.scripting.api.ScHwAnalysisConfig.isOwnershipTrace()}
\]

8.8.2.5 setTraceScenario

It sets the trace scenario by index.

\[
\text{void com.freescale.sa.sc.scripting.api.ScHwAnalysisConfig.setTraceScenario(TraceScenarioCode scenarioCode)}
\]

**Parameters**

**scenarioCode**

The TraceScenarioCode used to identify the trace scenario. It is based on one of the scenario codes provided by the com.freescale.sa.sc.launch.ui# ScHwTargetSettingsManager# TraceScenarioCode enum. The members of this enum are: PROFILING_CACHE_EVENTS, PROFILING_DATA_TRACE, PROFILING_CLOCK_CYCLES, PROFILING_ADVANCED, PROGRAM_TRACE, COVERAGE, NONE

8.8.2.6 getTraceScenario

It gets index of trace scenario set as default.

\[
\text{TraceScenarioCode com.freescale.sa.sc.scripting.api.ScHwAnalysisConfig.getTraceScenario()}
\]

**Returns**

The trace scenario based on one of the scenario codes provided by the com.freescale.sa.sc.launch.ui# ScHwTargetSettingsManager# TraceScenarioCode enum. The members of this enum are: PROFILING_CACHE_EVENTS, PROFILING_DATA_TRACE, PROFILING_CLOCK_CYCLES, PROFILING_ADVANCED, PROGRAM_TRACE, COVERAGE, NONE

8.8.2.7 load
Loads configuration using a launch config.

Creates a trace config file in the process (if required).

```java
void com.freescale.sa.sc.scripting.api.ScHwAnalysisConfig.load(ILaunchConfiguration launchConfig)
```

**Parameters**

`launchConfig`

name of the launch configuration.

**CoreException**

8.9  **com::freescale::sa::sc::scripting::api::ScSimAnalysisConfig**

Refer below.

8.9.1  **Public Member Functions**

Lists the public member functions of the `com::freescale::sa::sc::scripting::api::ScSimAnalysisConfig` class.

Following public members are available:

- `ScSimAnalysisConfig`
- `ScSimAnalysisConfig`
- `setAttribute`
- `getAttribute`

Comprehensive description of each of the above listed functions is as follows:

8.9.1.1  **ScSimAnalysisConfig**

```java
com.freescale.sa.sc.scripting.api.ScSimAnalysisConfig.ScSimAnalysisConfig(ILaunchConfiguration launchConfig)
```

8.9.1.2  **ScSimAnalysisConfig**
8.9.1.3 setAttribute

Set one of the SA StarCore simulator config attributes.

```java
void com.freescale.sa.sc.scripting.api.ScSimAnalysisConfig.setAttribute(Object attributeType, Object arg)
```

**Parameters**

- **attributeType**
  
  One of the ScSimAnalysisConfigAttributes enum attributes (COM_PORT_NUMBER, LOGGING, OUTPUT_FOLDER).

- **arg**
  
  The value the attribute will take.

8.9.1.4 getAttribute

Get the value of a specific SA StarCore simulator config attribute.

```java
Object com.freescale.sa.sc.scripting.api.ScSimAnalysisConfig.getAttribute(Object attributeType)
```

**Parameters**

- **attributeType**
  
  The attribute from which the value will be retrieved.

**Returns**

the specific attribute value.

8.9.2 Protected Member Functions

Lists the protected member functions of the
com::freescale::sa::sc::scripting::api::ScSimAnalysisConfig class.

Following protected members are available:
• load
• setCommPortNumber
• getCommPortNumber
• setOutputFolder
• enableLogging
• isLoggingEnabled

Comprehensive description of each of the above listed functions is as follows:

8.9.2.1  load

Loads configuration using a launch config.
Creates a trace config file in the process (if required).

```java
void com.freescale.sa.sc.scripting.api.ScSimAnalysisConfig.load(ILaunchConfiguration launchConfig)
```

**Parameters**

- `launchConfig`
  - name of the launch configuration.

  `CoreException`

8.9.2.2  setCommPortNumber

Sets the communication port.

```java
void com.freescale.sa.sc.scripting.api.ScSimAnalysisConfig.setCommPortNumber(int commPortNumber)
```

**Parameters**

- `commPortNumber`

8.9.2.3  getCommPortNumber

```java
int com.freescale.sa.sc.scripting.api.ScSimAnalysisConfig.getCommPortNumber()
```

**Returns**
The stored communication port number

8.9.2.4  `setOutputFolder`

Sets the results output folder.

```java
void com.freescale.sa.sc.scripting.api.ScSimAnalysisConfig.setOutputFolder(String outputFolder)
```

**Parameters**

`outputFolder`

the path to the new results folder

8.9.2.5  `enableLogging`

Enables logging.

```java
void com.freescale.sa.sc.scripting.api.ScSimAnalysisConfig.enableLogging(boolean enableLog)
```

**Parameters**

`enableLog`

Enable/disable logging.

8.9.2.6  `isLoggingEnabled`

Returns the current state of logging state.

```java
boolean com.freescale.sa.sc.scripting.api.ScSimAnalysisConfig.isLoggingEnabled()
```

**Returns**

true/false based on logging state.

8.10  `com::freescale::sa::scripting::api::TracePoint`

Refer below.
8.10.1 Protected Member Functions

Lists the protected member functions of the com::freescale::sa::scripting::api::TracePoint class.

Following protected members are available:

- TracePoint
- getITracepoint

Comprehensive description of each of the above listed functions is as follows:

8.10.1.1 TracePoint

com.freescale.sa.scripting.api.TracePoint.TracePoint(ITracepoint _tp)

8.10.1.2 getITracepoint

ITracepoint com.freescale.sa.scripting.api.TracePoint.getITracepoint()

8.10.2 Public Member Functions

Lists the public member functions of the com::freescale::sa::scripting::api::TracePoint class.

Following public members are available:

- setEnabled
- getEnabled
- getAction
- getType
- getAddress
- getFileName
- getLineNumber

Comprehensive description of each of the above listed functions is as follows:
8.10.2.1  setEnabled

Set enable state of the tracepoint.

```java
void com.freescale.sa.scripting.api.TracePoint.setEnabled(boolean en)
```

**Parameters**

<table>
<thead>
<tr>
<th>en</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable if true, disable if false.</td>
</tr>
</tbody>
</table>

8.10.2.2  getEnabled

Get enable state of the tracepoint.

```java
boolean com.freescale.sa.scripting.api.TracePoint.getEnabled()
```

**Returns**

True if tracepoint is enabled, false otherwise.

8.10.2.3  getAction

```java
Action com.freescale.sa.scripting.api.TracePoint.getAction()
```

**Returns**

The Action associated with the Tracepoint.

8.10.2.4  getType

```java
Type com.freescale.sa.scripting.api.TracePoint.getType()
```

**Returns**

The Type of the tracepoint.
8.10.2.5  getAddress

Get address of tracepoint.

String com.freescale.sa.scripting.api.TracePoint.getAddress()

Returns
The address of the tracepoint.

8.10.2.6  getFileName

Get source file name of the tracepoint.

String com.freescale.sa.scripting.api.TracePoint.getFileName()

Returns
The source file name for the tracepoint.

8.10.2.7  getLineNumber

Get line number of tracepoint.

int com.freescale.sa.scripting.api.TracePoint.getLineNumber()

Returns
The line number of the tracepoint.

8.11  com.freescale.sa.scripting.IAnalysisConfigFactory

Refer below.

8.11.1  Public Member Functions

Lists the public member functions of the
com.freescale.sa.scripting.IAnalysisConfigFactory interface.
Following public static member functions are available:

- `createAnalysisConfigFromArchive`
- `createAnalysisConfigFromLaunch`

### 8.11.1.1 createAnalysisConfigFromArchive

Creates an AnalysisConfig object from a trace config file.

```java
AnalysisConfig
com.freescale.sa.scripting.IAnalysisConfigFactory.createAnalysisConfigFromArchive(String configPath)
```

**Parameters**

`configPath`

The path of the configuration file archive.

**Returns**

An object. AnalysisConfig

### 8.11.1.2 createAnalysisConfigFromLaunch

Creates an AnalysisConfig object for a launch config name. May create a trace config file in the process (if needed).

```java
AnalysisConfig
com.freescale.sa.scripting.IAnalysisConfigFactory.createAnalysisConfigFromLaunch(String launchConfigName)
```

**Parameters**

`launchConfigName`

The launch configuration name.

**Returns**

An object. AnalysisConfig
Index

A
About CodeWarrior Interface 15
About Data Collection 17
About this Guide 13
Accompanying Documentation 14
addAddressTracePoint 194
Adding Connection Manually 165
Add or Remove Function 62
Add or Remove Group 62
addSrcLineTracePoint 194
Add Stock Metrics 181
AnalysisActions 190
AnalysisConfig 192
AnalysisResults 204

C
Call Tree Data 18
Call Tree View 87
Change Color 62
Change Trace Configuration 93
Collecting Data 21
Collecting Multicore Trace Data 114
Collecting Trace 46
Collect Trace Using Jython 186
Configuration View 153
Configuring Launcher 33
Configuring Time Unit and Time Format 58
Connecting to Board 165
Connection Pane 154
Counterpoints 18
Count events if PC is in Address 0..Address 1 Range 158
Count events unconditionally 158
createAnalysisConfigFromArchive 222
createAnalysisConfigFromLaunch 222
Create an Attach Launch Configuration 89, 93
Creating New Configuration 162
Creating New Project 22
Critical Code Data 18
Critical Code View 85
Custom Scenario Editor
  Events Tab 178
  Initialization Tab 181
  Metrics Tab 179
  Overview Tab 177
  Reports Tab 182
Custom scenarios 176

D
Debugging Multicore Project 110
Defining custom scenarios 176
Disassembly View 135

E
Edit Address Range of Function 62
Editor Area 136
enableLogging 218
Enabling and Disabling the Counterpoints 150
Enabling and Disabling the Tracepoints 138
Enabling Aurora 95
Exclude Symbols from Statistics 80
Executing Aurora Training Script 95
Export Critical Code Data 69
exportFunctionResults 206
Exporting Trace Data 57
exportResults 205
exportTraceData 205

F
Filter Core Events pane 157
Filter reference clock event 158

G
getAction 220
getAddress 221
getAnalysisConfig 201
getAnalysisConfigFromLaunch 201
getAnalysisResults 201–203
getAttribute 198, 208, 212, 216
getCommPortNumber 217
getEnabled 220
getAddress 221
getIgnoreAllTracePoints 196
getITracepoint 219
getLaunchConfiguration 193
getLineNumber 221
getOutputFolder 193
Getting Started 15
getTracePoints 195
getTraceRecordsNo 205
getTraceScenario 214
getType 220
getAddress 221
getWorkingCopy 192
Graphical Format 173

H
Hardware Profiling 27, 38
Using Simulator 47

V

Viewing Call Tree Data 76
Viewing Counterpoints 77, 80
Viewing Counterpoints in Analysispoints 149
Viewing Critical Code Data 65, 80
Viewing Data 49
Viewing Data Generated by Hardware 77
Viewing Data Generated by Simulator 50
Viewing Interrupts 93
Viewing multicore trace data 114
Viewing Performance Data 72, 80
Viewing Results 171
Viewing Timeline Data 59, 79
Viewing Trace Data 50, 78
Viewing Tracepoints in Analysispoints 135